IX. Mathematical Language (October 2008)

This chapter contains the definition of the *Mathematical Language* we use in this book. It is made of seven sections. The first one contains a preliminary definition of sequents, inference rules, and proofs. Then we have the presentation of our Mathematical Language. It is defined as follows: the Propositional Language (section 2), the Predicate Language (section 3), the Equality Language (section 4), the Set-theoretic Language (section 5), and the Boolean and Arithmetic Language (section 6). Each of these languages will be presented as an extension of the previous one. A final section contains a definition of various data structures which we are going to use in subsequent chapters among which are lists, rings and trees.

1 Sequent Calculus

1.1 Definitions

In this section, we give some definitions which will be helpful to present the Sequent Calculus.

(1) A *sequent* is a generic name for "something we want to prove". For the moment, this is just an informally defined notion, which we shall refine later in section 1.2. The important thing to note at this point is that we can associate a *proof* with a sequent. For the moment, we do not know what a proof is however. It will only be defined at the end of this section.

(2) An *inference rule* is a device used to construct proofs of sequents. It is made of two parts: the *antecedent* part and the *consequent* part. The antecedent denotes a finite set of sequents while the consequent denotes a single sequent. An inference rule, named say **R1**, with antecedent A and consequent C is usually written as follows:

$$\frac{\mathsf{A}}{C}\ \textbf{R1}$$

It is to be read:

> Inference Rule **R1** yields a proof of sequent C as soon as we have proofs of each sequent of A.

The antecedent A might be empty. In this case, the inference rule, named say **R2**, is written as follows:

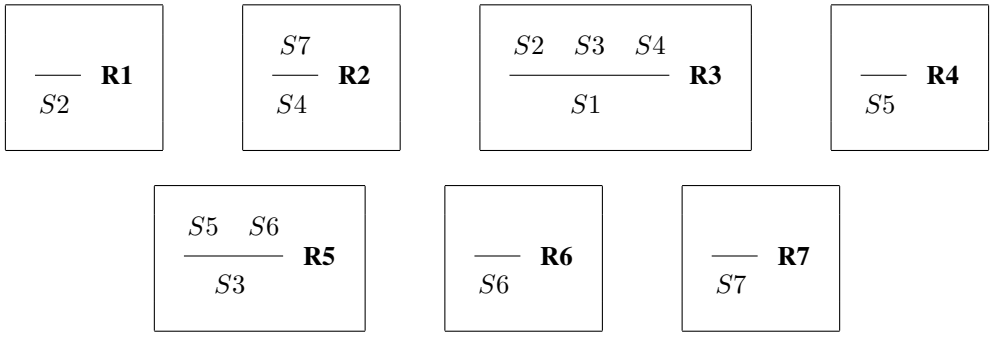
$$\frac{}{C}\ \textbf{R2}$$

It is to be read:

> Inference Rule **R2** yields a proof of sequent C.

(3) A *theory* is a set of inference rules.

(4) The *proof of a sequent* within a theory is simply a finite tree with certain constraints. The nodes of such a tree have two components: a sequent s and a rule r of the theory. Here are the constraints for each node of the form (s, r): the consequent of the rule r is s, and the children of this node are nodes whose sequents are exactly all the sequents of the antecedent of rule r. As a consequence, the leaves of the tree contain rules with no antecedent. Moreover, the root node of the tree contains the sequent to be proved. As an example, let be given the following theory involving sequents $S1$ to $S7$ and rules **R1** to **R7**:



On figure 1 you can see a proof of sequent $S1$:

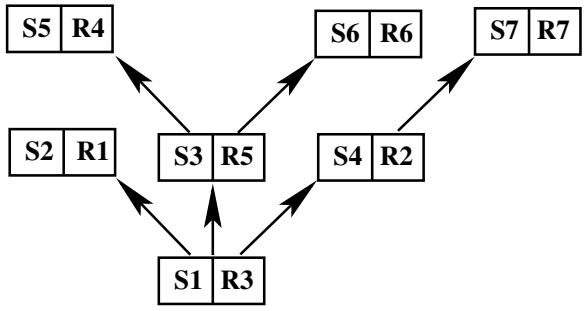


Fig. 1. A Proof

As can be seen, the root of the tree contains sequent $S1$, which is the one we want to prove. And it is easy to check that each node, say node $(S3, \textbf{R5})$, is indeed such that the consequent of its rule is the sequent of the node. More precisely, $S3$ in this case, is the consequent of rule **R5**. Moreover, we can check that the sequents of the child nodes of node $(S3, \textbf{R5})$, namely, $S5$ and $S6$, are exactly the sequents forming the antecedents of rule **R5**.

This tree can be interpreted as follows: In order to prove $S1$, we prove $S2$, $S3$, and $S4$, according to rule **R3**. In order to prove $S2$ we prove nothing more, according to rule **R1**. In order to prove $S3$ we prove $S5$ and $S6$, according to **R5**. And so on.

This tree can be represented as we have done in chapter 2: this is indicated on figure 2. In this chapter, we shall adopt this representation as well.

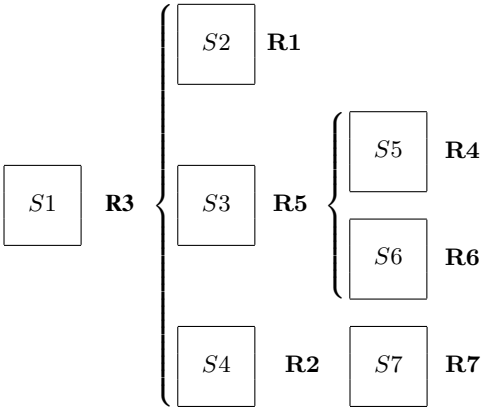


Fig. 2. Another Representation of the Proof Tree

1.2 Sequents for a Mathematical Language

We now refine our notion of sequent in order to define the way we shall make proofs with our Mathematical Language. Such a language contains constructs called *Predicates*. For the moment, this is all what we know about our Mathematical Language. Within this framework, a sequent S, as defined in the previous section, now becomes a more complex object. It is made of two parts: the *hypotheses* part and the *goal* part. The hypothesis part denotes a finite set of predicates while the goal part denotes a single predicate. A sequent with hypotheses H and goal G is written as follows:

$$\mathsf{H} \vdash G$$

This sequent is to be read as follows:

Goal G holds under the set of hypotheses H

This is the sort of sequents we want to prove. It is also the sort of sequents we shall have in the theories associated with our Mathematical Language. Note that the set of hypotheses of a sequent might be empty and that the order and repetition of hypotheses in the set H is meaningless.

1.3 Initial Theory

We now have enough elements at our disposal to define the first rules of our proving theory. Note again that we still don't know what a predicate is. We just know that predicates are constructs we shall be able to define within our future Mathematical Language. We start with three basic rules which we first state informally and then define more rigorously. They are called **HYP**, **MON**, and **CUT**. Here are their definitions:

– **HYP**: If the goal P of a sequent belongs to the set of hypotheses of this sequent, then it is proved.

$$\frac{}{\mathsf{H},\ P\ \vdash\ P}\ \textbf{HYP}$$

– **MON**: In order to prove a sequent, it is sufficient to prove another sequent with the same goal but with less hypotheses.

$$\frac{\mathsf{H}\ \vdash\ Q}{\mathsf{H},\ P\ \vdash\ Q}\ \textbf{MON}$$

– **CUT**: If you succeed in proving a predicate P under a set of hypotheses H, then P can be added to the set of hypotheses H for proving a goal Q.

$$\frac{\mathsf{H}\ \vdash\ P \qquad \mathsf{H},\ P\ \vdash\ Q}{\mathsf{H}\ \vdash\ Q}\ \textbf{CUT}$$

Note that in the previous rules, the letter H, P and Q are, so-called, *meta-variables*. The letter H is a meta-variable standing for a finite set of predicates, whereas the letter P and Q are meta-variables standing for predicates. Clearly then each of the previous "rules" stands for more than just one rule: it is better to call it a *rule schema*. This will always be the case in what follows.

2 The Propositional Language

In this section we present a first simple version of our Mathematical Language, it is called the Propositional Language. It will be later refined to more complete versions: Predicate Language (section 3), Equality Language (section 4), Set-theoretic Language (section 5), and Arithmetic language (section 6).

2.1 Syntax

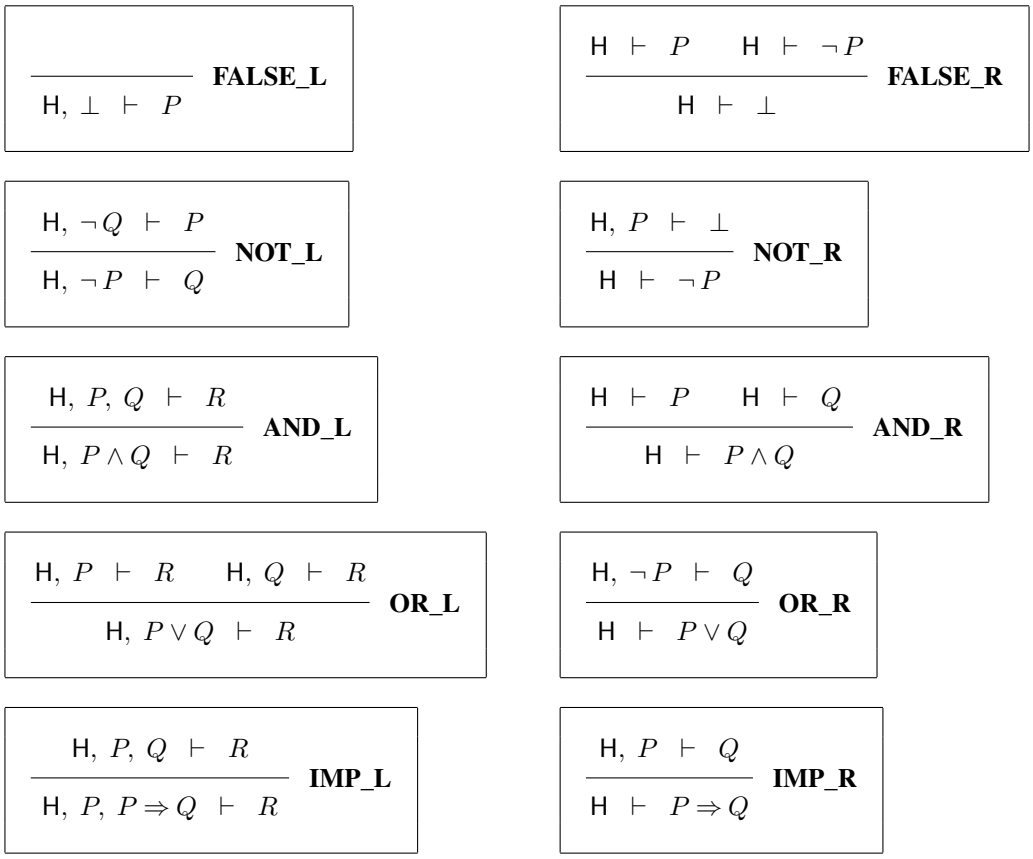
Our first version is built around five constructs called *falsity*, *negation*, *conjunction*, *disjunction*, and *implication*. Given two predicates P and Q, we can construct their conjunction $P \wedge Q$, their disjunction $P \vee Q$, and their implication $P \Rightarrow Q$. And given a predicate P, we can construct its negation $\neg P$. This can be formalized by means of the following syntax:

$$\begin{array}{rcl} \mathit{predicate} & ::= & \bot \\ & & \neg\, \mathit{predicate} \\ & & \mathit{predicate}\ \wedge\ \mathit{predicate} \\ & & \mathit{predicate}\ \vee\ \mathit{predicate} \\ & & \mathit{predicate}\ \Rightarrow\ \mathit{predicate} \end{array}$$

This syntax is clearly ambiguous, but we do not care about it at this stage. Only note that conjunction and disjunction operators have stronger syntactic priorities than the implication operator. Moreover, conjunction and disjunction have the same syntactic priorities so that parentheses will always be necessary when several such distinct operators are following each other. Also note that this syntax does not contain any "base" predicate (except $\bot$): such predicates will come later in sections 4 and 5.

2.2 Enlarging the Initial Theory

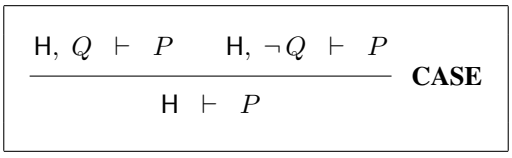
The initial theory of section 1.3 is enlarged with the following inference rules:



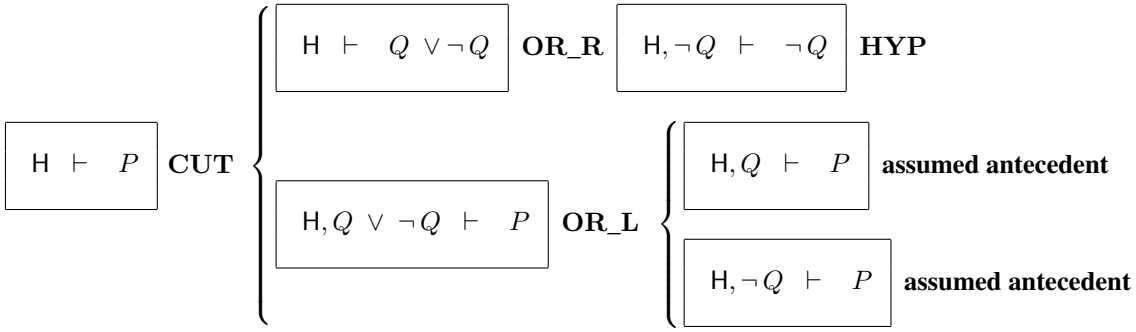
As can be seen, each kind of predicates, namely falsity, negation, conjunction, disjunction, and implication, is given two rules: a left rule, labelled with **_L**, and a right rule, labelled with **_R**. This corresponds to the predicate appearing either in the hypothesis part (left) or in the goal part (right) of the consequent of the rule.

2.3 Derived Rules

Besides the previous rules the following *derived* rule (among many others) is quite useful. It says that for proving a goal P it is sufficient to prove it first under hypothesis Q and then under hypothesis $\neg Q$.



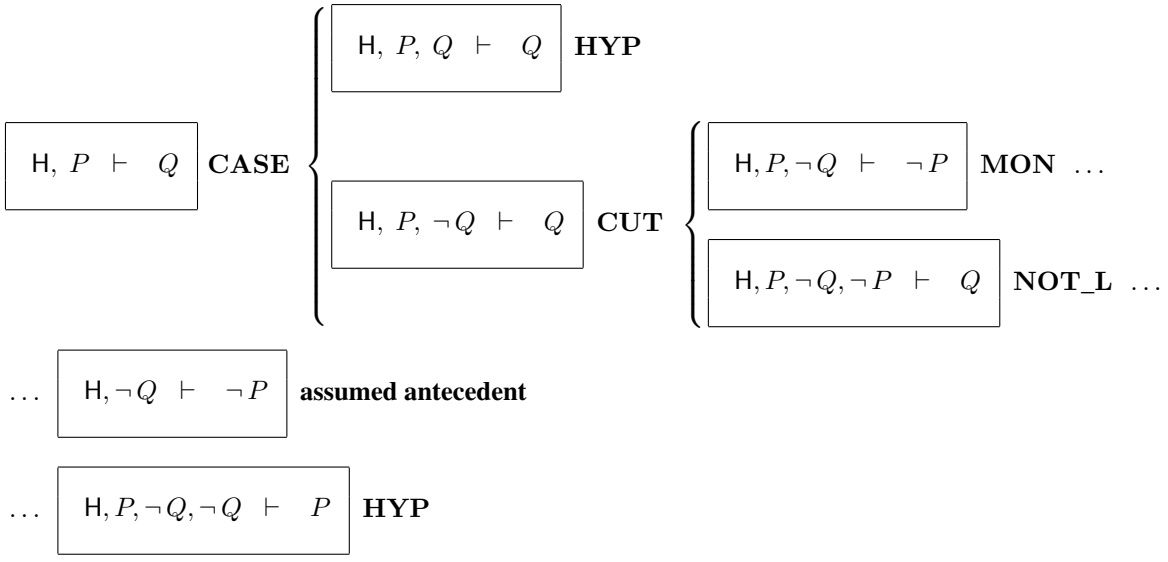
For proving a derived rule, we assume its antecedents (if any) and prove its consequent. With this in mind, here is the proof of derived rule **CASE**:



With the help of this new (derived) rule **CASE**, we can now generalize rule **NOT_L** by rule **CT_L**:

$$\frac{\mathsf{H}.\ \neg Q \ \vdash\ \neg P}{\mathsf{H},\ P \ \vdash\ Q}\ \textbf{CT\_L}$$

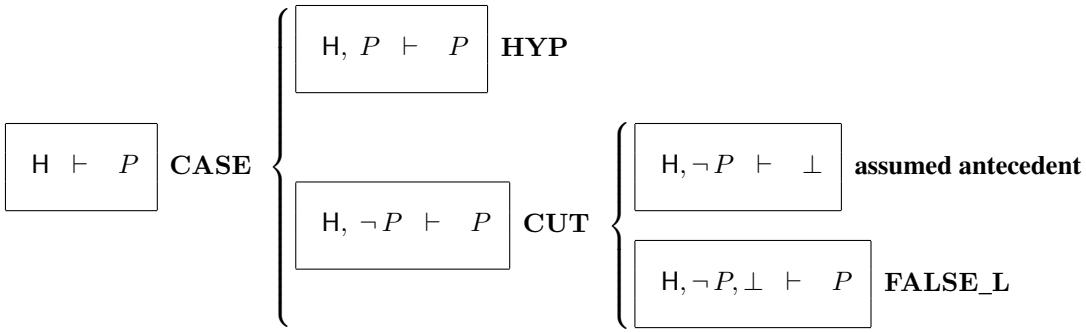
Proof of rule **CT_L**:



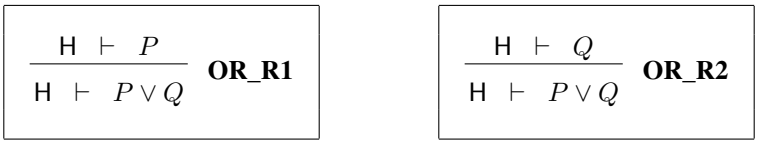
We can also generalize rule **NOT_R** by rule **CT_R**

$$\frac{\mathsf{H},\ \neg P \ \vdash\ \bot}{\mathsf{H} \ \vdash\ P}\ \textbf{CT\_R}$$

Proof of rule **CT_R**:



In a similar way, we can prove the following derived rules, which we used in chapter 2.



2.4 Methodology

The method we are going to use to build our Mathematical Language must start to be clearer: it will be very systematic. It is made of two steps: first we augment our syntax. Then either the extension corresponds to a simple facility. In that case, we give simply the definition of the new construct in terms of previous ones. Or the new construct is not related to any previous constructs. In that case, we augment our current theory.

2.5 Extending the Proposition Language

The Proposition Language is now extended by adding one more construct called *equivalence*. Given two predicates P and Q, we can construct their equivalence $P \Leftrightarrow Q$. We also add one predicate: $\top$. As a consequence, our syntax is now the following:

$$
\begin{array}{rcl}
predicate & ::= & \bot \\
 & & \top \\
 & & \neg\, predicate \\
 & & predicate \wedge predicate \\
 & & predicate \vee predicate \\
 & & predicate \Rightarrow predicate \\
 & & predicate \Leftrightarrow predicate
\end{array}
$$

Note that implication and equivalence operators have the same syntactic priorities so that parentheses will be necessary when several such distinct operators are following each other. Such extensions are defined in terms of previous ones by mere rewriting rules:

Predicate	Rewritten
$\top$	$\neg\ \bot$
$P\ \Leftrightarrow\ Q$	$(P\ \Rightarrow\ Q)\ \wedge\ (Q\ \Rightarrow\ P)$

The following derived rules can be proved easily:

$$\frac{\mathsf{H}\ \vdash\ P}{\mathsf{H},\ \top\ \vdash\ P}\ \textbf{TRUE\_L} \qquad\qquad \frac{}{\mathsf{H}\ \vdash\ \top}\ \textbf{TRUE\_R}$$

Note that rule **TRUE_L** can be proved using rule **MON** but the reverse rule (exchanging antecedent and consequent), which holds as well, cannot. We leave it as an exercise to the reader to prove these rules.

3 The Predicate Language

3.1 Syntax

In this section, we introduce the Predicate Language. The syntax is extended with a number of new kinds of predicates and also with the introduction of two new syntactic categories called *expression* and *variable*. A *variable* is a simple identifier. Given a non-empty list of variables x made of pairwise distinct identifiers and a predicate P, the construct $\forall x \cdot P$ is called a *universally quantified predicate*. Likewise, given a non-empty list of variables x made of pairwise distinct identifiers and a predicate P, the construct $\exists x \cdot P$ is called an *existentially quantified predicate*. An *expression* is either a variable or else a *paired expression* $E \mapsto F$, where E and F are two expressions. Here is this new syntax:

$$\begin{array}{lll}
predicate & ::= & \bot \\
 & & \top \\
 & & \neg\, predicate \\
 & & predicate\ \wedge\ predicate \\
 & & predicate\ \vee\ predicate \\
 & & predicate\ \Rightarrow\ predicate \\
 & & predicate\ \Leftrightarrow\ predicate \\
 & & \forall var_list \cdot predicate \\
 & & \exists var_list \cdot predicate \\
 & & \\
expression & ::= & variable \\
 & & expression \mapsto expression \\
 & & \\
var_list & ::= & variable \\
 & & variable,\ var_list
\end{array}$$

This syntax is also ambiguous. Note however that the scope of the universal or existential quantifiers extends to the right as much as they can, the limitation being expressed either by the end of the formula or by means of enclosing parentheses.

3.2 Predicates and Expressions

It might be useful at this point to clarify the difference between a predicate and an expression. A predicate P is a piece of formal text which can be *proved* when embedded within a sequent as in:

$$\mathrm{H} \vdash P$$

A predicate does not denote anything. This is not the case of an expression which always denotes an *object*. An expression cannot be "proved". Hence predicates and expressions are incompatible. Note that for the moment the possible expressions we can define are quite limited. This will be considerably extended in the Set-theoretic Language defined in Section 5.

3.3 Inference Rules for Universally Quantified Predicates

The universally and existentially quantified predicates require introducing corresponding rules of inference. As for propositional calculus, in both cases we need two rules: one for quantified assumptions (left rule) and one for a quantified goal (right rule). Here are these rules for universally quantified predicates:

$$\frac{\mathrm{H},\ \forall x \cdot P,\ [x := E]P \ \vdash\ Q}{\mathrm{H},\ \forall x \cdot P \ \vdash\ Q}$$ **ALL_L**

$$\frac{\mathrm{H} \ \vdash\ P}{\mathrm{H} \ \vdash\ \forall x \cdot P}$$ **ALL_R** (x not free in H)

The first rule (ALL_L) allows us to add another assumption when we have a universally quantified one. This new assumption is obtained by instantiating the quantified variable x by any expression E in the predicate P: this is denoted by $[x := E]P$. The second rule (ALL_R) allows us to remove the "$\forall$" quantifier appearing in the goal. This can be done however only if the quantified variable (here x) *does not appear free* in the the set of assumptions H: this requirement is called a *side condition*. In the sequel we shall write x $\underline{\mathsf{nfin}}$ P to mean that variable x is not free in predicate P. The same notation is used with an expression E. We omit in this presentation to develop the syntactic rules allowing us to compute non-freeness as well as substitutions. We have similar rules for existentially quantified predicates:

$$\frac{\mathrm{H},\ P \ \vdash\ Q}{\mathrm{H},\ \exists x \cdot P \ \vdash\ Q}$$ **XST_L** (x not free in H and Q)

$$\frac{\mathrm{H} \ \vdash\ [x := E]P}{\mathrm{H} \ \vdash\ \exists x \cdot P}$$ **XST_R**

As an example, we prove now the following sequent:

$$\forall x \cdot (\exists y \cdot P_{x,y}) \ \Rightarrow\ Q_x \quad \vdash \quad \forall x \cdot (\forall y \cdot P_{x,y} \ \Rightarrow\ Q_x)$$

where $P_{x,y}$ stands for a predicate containing variables x and y only as free variables, and Q_x stands for a predicate containing variable x only as a free variable.

$$\frac{\dfrac{\dfrac{}{\forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x \vdash \forall x \cdot (\forall y \cdot P_{x,y} \Rightarrow Q_x)}\ \mathsf{ALL\_R}\ \mathsf{ALL\_R}\ \mathsf{IMP\_R} \qquad \dfrac{}{\forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x,\ P_{x,y} \vdash Q_x}\ \mathsf{CUT}\ \ldots}{\ldots \quad \left\{ \begin{array}{l} \dfrac{}{\forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x,\ P_{x,y} \vdash \exists y \cdot P_{x,y}}\ \mathsf{XST\_R} \quad \dfrac{}{\forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x,\ P_{x,y} \vdash P_{x,y}}\ \mathsf{HYP} \\ \dfrac{}{\forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x,\ P_{x,y},\ \exists y \cdot P_{x,y} \vdash Q_x}\ \mathsf{ALL\_L}\ \mathsf{IMP\_L} \quad \dfrac{}{\forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x,\ Q_x,\ P_{x,y},\ \exists y \cdot P_{x,y} \vdash Q_x}\ \mathsf{HYP} \end{array} \right.}$$

The proof of the following sequent is left to the reader:

$$\forall x \cdot (\forall y \cdot P_{x,y} \Rightarrow Q_x) \quad \vdash \quad \forall x \cdot (\exists y \cdot P_{x,y}) \Rightarrow Q_x$$

An interesting derived rule is the following, which allows one to simplify an existential goal by replacing it by another one, hopefully simpler:

$$\frac{\mathsf{H} \vdash \exists x \cdot Q \qquad \mathsf{H},\ Q \vdash P}{\mathsf{H} \vdash \exists x \cdot P}\ \textbf{CUT\_XST}\ (\text{x } \underline{\text{nfin}}\ \mathsf{H})$$

Proof of **CUT_XST**

$$\frac{\mathsf{H} \vdash \exists x \cdot P}{\left\{ \begin{array}{l} \mathsf{H} \vdash \exists x \cdot Q \quad \textbf{assumed antecedent} \\ \dfrac{\mathsf{H},\ \exists x \cdot Q \vdash \exists x \cdot P}{\mathsf{H},\ Q \vdash P}\ \begin{array}{l}\textbf{XST\_L}\\ \textbf{XST\_R}\end{array} \quad \textbf{assumed antecedent} \end{array} \right.}\ \textbf{CUT}$$

4 Introducing Equality

The Predicate Language is once again extended by adding a new predicate, the *equality predicate*. Given two expressions E and F, we define their equality by means of the construct $E = F$. Here is the extension of our syntax:

$$
\begin{array}{lll}
predicate & ::= & \bot \\
 & & \top \\
 & & \neg\, predicate \\
 & & predicate \ \wedge \ predicate \\
 & & predicate \ \vee \ predicate \\
 & & predicate \ \Rightarrow \ predicate \\
 & & predicate \ \Leftrightarrow \ predicate \\
 & & \forall var\_list \cdot predicate \\
 & & \exists var\_list \cdot predicate \\
 & & expression = expression \\
 & & \\
expression & ::= & variable \\
 & & expression \mapsto expression
\end{array}
$$

Note that we shall use the operator $\neq$ in the sequel to mean, as is usual, the negation of equality. The inference rules for equality are the following:

$$\frac{[x := F]\mathrm{H},\ E = F \ \vdash \ [x := F]P}{[x := E]\mathrm{H},\ E = F \ \vdash \ [x := E]P} \quad \textbf{EQ\_LR}$$

$$\frac{[x := E]\mathrm{H},\ E = F \ \vdash \ [x := E]P}{[x := F]\mathrm{H},\ E = F \ \vdash \ [x := F]P} \quad \textbf{EQ\_RL}$$

It allows us to *apply* an equality assumption in the remaining assumptions and in the goal. This can be made by using the equality from left to right or from right to left. Subsequent rules correspond to the reflexivity of equality and to the equality of pairs. They are both defined by some rewriting rules as follows:

Operator	Predicate	Rewritten
Equality	$E = E$	$\top$
Equality of pairs	$E \mapsto F \ = \ G \mapsto H$	$E = G \ \wedge \ F = H$

The following rewriting rules, within which x is supposed to be not free in E, are easy to prove. They are called the *one point rules*:

Predicate	Rewritten
$\forall x \cdot x = E \ \Rightarrow \ P$	$[x := E]P$
$\exists x \cdot x = E \ \wedge \ P$	$[x := E]P$

5 The Set-theoretic Language

Our next language, the Set-theoretic Language, is now presented as an extension to the previous Predicate Language.

5.1 Syntax

In this extension, we introduce some special kind of expressions called *sets*. Note that not all expressions are set: for instance a pair is not a set. However, in the coming syntax we shall not make any distinction between expressions which are sets and expressions which are not.

We introduce another predicate the *membership predicate*. Given an expression E and a set S, the construct $E \in S$ is a membership predicate which says that expression E is a *member* of set S.

We also introduce the basic set constructs. Given two sets S and T, the construct $S \times T$ is a set called the *Cartesian product* of S and T. Given a set S, the construct $\mathbb{P}(S)$ is a set called the *power set of* S. Finally, given a list of variables x with pairwise distinct identifiers, a predicate P, and an expression E, the construct $\{x \cdot P \mid E\}$ is called a *set defined in comprehension*. Here is our new syntax:

$$
\begin{array}{lll}
predicate & ::= & \ldots \\
 & & expression \ \in \ expression \\
 & & \\
expression & ::= & variable \\
 & & expression \mapsto expression \\
 & & expression \times expression \\
 & & \mathbb{P}(expression) \\
 & & \{\, var\_list \cdot predicate \mid expression \,\}
\end{array}
$$

Note that we shall use the operator $\notin$ in the sequel to mean, as is usual, the negation of set membership.

5.2 Axioms of Set Theory

The axioms of the set-theoretic Language are given under the form of equivalences to various set memberships. They are all defined in terms of rewriting rules. Note that the last of these rules defines equality for sets. It is called the *Extensionality Axiom*.

Operator	Predicate	Rewritten	Side Cond.
Cartesian product	$E \mapsto F \in S \times T$	$E \in S \ \wedge \ F \in T$	
Power set	$E \in \mathbb{P}(S)$	$\forall x \cdot x \in E \ \Rightarrow \ x \in S$	$x \ \underline{\mathsf{nfin}} \ E$ $x \ \underline{\mathsf{nfin}} \ S$
Set comprehension	$E \in \{ x \cdot P \mid F \}$	$\exists x \cdot P \ \wedge \ E = F$	$x \ \underline{\mathsf{nfin}} \ E$
Set equality	$S = T$	$S \in \mathbb{P}(T) \ \wedge \ T \in \mathbb{P}(S)$	

As a special case, set comprehension can sometimes be written $\{ F \mid P \}$, which can be read as follows: "the set of objects with shape F when P holds". However, as you can see, the list of variables x has now disappeared. In fact, these variables are then *implicitly determined* as being all the free variables in F. When we want that x represent only *some*, but not all, of these free variables we cannot use this shorthand.

A more special case is one where the expression F is exactly a single variable x, that is $\{ x \cdot P \mid x \}$. As a shorthand, this can be written $\{ x \mid P \}$, which is very common in informally written mathematics. And then $E \in \{ x \mid P \}$ becomes $[x := E]P$ according to the second "one point rule" of section 4.

5.3 Elementary Set Operators

In this section, we introduce the classical set operators: inclusion, union, intersection, difference, extension, and the empty set.

$$
\begin{array}{lll}
\mathit{predicate} & ::= & \ldots \\
 & & \mathit{expression} \subseteq \mathit{expression} \\
 & & \\
\mathit{expression} & ::= & \ldots \\
 & & \mathit{expression} \cup \mathit{expression} \\
 & & \mathit{expression} \cap \mathit{expression} \\
 & & \mathit{expression} \setminus \mathit{expression} \\
 & & \{\mathit{expression\_list}\} \\
 & & \varnothing \\
 & & \\
\mathit{expression\_list} & ::= & \mathit{expression} \\
 & & \mathit{expression},\ \mathit{expression\_list}
\end{array}
$$

Notice that the expressions in an $\mathit{expression\_list}$ are not necessarily distinct.

Operator	Predicate	Rewritten
Inclusion	$S \subseteq T$	$S \in \mathbb{P}(T)$
Union	$E \in S \cup T$	$E \in S \ \lor \ E \in T$
Intersection	$E \in S \cap T$	$E \in S \ \land \ E \in T$
Difference	$E \in S \setminus T$	$E \in S \ \land \ \neg (E \in T)$
Set extension	$E \in \{a, \ldots, b\}$	$E = a \ \lor \ \ldots \ \lor \ E = b$
Empty set	$E \in \varnothing$	$\bot$

5.4 Generalization of Elementary Set Operators

The next series of operators consists in generalizing union and intersection to sets of sets. This takes the forms either of an operator acting on a set or of a quantifier.

$$
\begin{array}{l}
\ldots \\
\\
expression \ ::= \ \ldots \\
\qquad\qquad \mathrm{union}(expression) \\
\qquad\qquad \bigcup var\_list \cdot predicate \mid expression \\
\qquad\qquad \mathrm{inter}(expression) \\
\qquad\qquad \bigcap var\_list \cdot predicate \mid expression
\end{array}
$$

Operator	Predicate	Rewritten	Side Cond.
Generalized intersection	$E \ \in \ \mathsf{union}\,(S)$	$\exists s \cdot s \in S \ \land \ E \in s$	$s \ \underline{\mathsf{nfin}} \ S$ $s \ \underline{\mathsf{nfin}} \ E$
Quantified union	$E \ \in \ \bigcup \ x \cdot P \mid T$	$\exists x \cdot P \ \land \ E \in T$	$x \ \underline{\mathsf{nfin}} \ E$

Operator	Predicate	Rewritten	Side Cond.
Generalized intersection	$E \in \mathrm{inter}(S)$	$\forall s \cdot s \in S \Rightarrow E \in s$	s nfin S s nfin E
Quantified intersection	$E \in \bigcap x \cdot P \mid T$	$\forall x \cdot P \Rightarrow E \in T$	x nfin E

The last two rewriting rules require that the set $\mathrm{inter}(S)$ and $\bigcap x \cdot P \mid T$ be *well defined*. This is presented in the following table:

Set construction	Well-definedness condition
$\mathrm{inter}(S)$	$S \neq \varnothing$
$\bigcap x \cdot P \mid T$	$\exists x \cdot P$

Well definedness conditions are taken care in proof obligations as explained in Section 2.12 of Chapter 5.

5.5 Binary Relation Operators

We now define a first series of binary relation operators: the set of binary relations built on two sets, the domain and range of a binary relation, and then various sets of binary relations.

$$
\begin{array}{l}
\ldots \\
\\
expression \ ::= \ \ldots \\
\qquad\qquad expression \leftrightarrow expression \\
\qquad\qquad \mathrm{dom}(expression) \\
\qquad\qquad \mathrm{ran}(expression) \\
\qquad\qquad expression \nleftrightarrow expression \\
\qquad\qquad expression \twoheadrightarrow expression \\
\qquad\qquad expression \nleftrightarrow expression
\end{array}
$$

Operator	Predicate	Rewritten	Side Cond.
Set of all binary relations	$r \in S \leftrightarrow T$	$r \subseteq S \times T$	
Domain	$E \in \mathrm{dom}(r)$	$\exists y \cdot E \mapsto y \in r$	y nfin E y nfin r

Operator	Predicate	Rewritten	Side Cond.
Range	$F \in \mathrm{ran}\,(r)$	$\exists x \cdot x \mapsto F \ \in \ r$	x $\underline{\mathsf{nfin}}$ F x $\underline{\mathsf{nfin}}$ r
Set of all total relations	$r \in S \nleftrightarrow T$	$r \in S \leftrightarrow T \ \wedge \ \mathrm{dom}\,(r) = S$	
Set of all surjective relations	$r \in S \leftrightarrow\!\!\!\!\rightarrow T$	$r \in S \leftrightarrow T \ \wedge \ \mathrm{ran}\,(r) = T$	
Set of all total and surjective relations	$r \in S \nleftrightarrow\!\!\!\!\rightarrow T$	$r \in S \nleftrightarrow T \ \wedge \ r \in S \leftrightarrow\!\!\!\!\rightarrow T$	

The next series of binary relation operators define the converse of a relation, various relation restrictions and the image of a set under a relation.

$$
\begin{array}{rl}
expression & ::= \ \ldots \\
& \quad expression^{-1} \\
& \quad expression \lhd expression \\
& \quad expression \rhd expression \\
& \quad expression \mathrel{\triangleleft\!\!\!\cdot} expression \\
& \quad expression \mathrel{\cdot\!\!\!\triangleright} expression \\
& \quad expression[expression]
\end{array}
$$

Operator	Predicate	Rewritten	Side Cond.
Converse	$E \mapsto F \in r^{-1}$	$F \mapsto E \in r$	
Domain restriction	$E \mapsto F \ \in \ S \lhd r$	$E \in S \ \ \wedge \ \ E \mapsto F \in r$	
Range restriction	$E \mapsto F \ \in \ r \rhd T$	$E \mapsto F \in r \ \ \wedge \ \ F \in T$	
Domain subtraction	$E \mapsto F \ \in \ S \mathrel{\triangleleft\!\!\!\cdot} r$	$\neg\, E \in S \ \ \wedge \ \ E \mapsto F \in r$	
Range subtraction	$E \mapsto F \ \in \ r \mathrel{\cdot\!\!\!\triangleright} T$	$E \mapsto F \in r \ \ \wedge \ \ \neg\, F \in T$	
Relational Image	$F \in r[U]$	$\exists x \cdot x \in U \ \wedge \ x \mapsto F \in r$	x $\underline{\mathsf{nfin}}$ F x $\underline{\mathsf{nfin}}$ r x $\underline{\mathsf{nfin}}$ U

Let us illustrate the relational image. Given a binary relation r from a set S to a set T, the image of a subset U of S under the relation r is a subset of T. The image of U under r is denoted by $r[U]$. Here is its definition:

$$r[U] \;=\; \{\, y \mid \exists x \cdot x \in U \;\wedge\; x \mapsto y \;\in\; r \,\}$$

This is illustrated on figure 3. As can be seen on this figure, the image of the set $\{a, b\}$ under relation r is the set $\{m, n, p\}$.

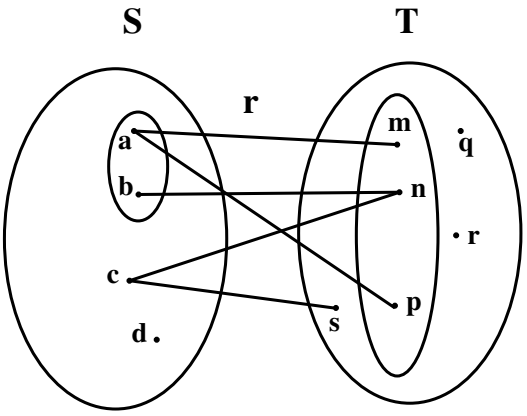


Fig. 3. Image of a Set under a Relation

Our next series of operators defines the composition of two binary relations, the overriding of a relation by another one, and the direct and parallel products of two relations.

$$\begin{array}{lll} expression & ::= & \ldots \\ & & expression \,;\, expression \\ & & expression \circ expression \\ & & expression \lhd\!\!\!+ expression \\ & & expression \otimes expression \\ & & expression \parallel expression \end{array}$$

Operator	Predicate	Rewritten	Side Cond.
Forward composition	$E \mapsto F \in f \,;\, g$	$\exists x \cdot E \mapsto x \in f \;\;\wedge\;\; x \mapsto F \in g$	x nfin E x nfin F x nfin f x nfin g
Backward composition	$E \mapsto F \in g \circ f$	$E \mapsto F \in f \,;\, g$	

Given a relation f from S to T and a relation g from T to U, the forward relational composition of f and g is a relation from S to U. It is denoted by the construct $f\ ;\ g$. Sometimes it is denoted the other way around as $g \circ f$, in which case is is said to be the backward composition. Figure 4 illustrates forward composition.

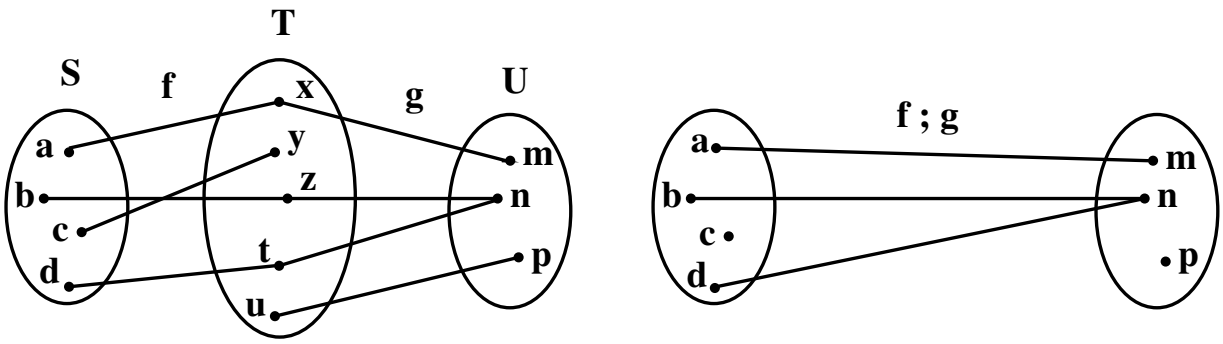


Fig. 4. Forward Composition

Operator	Predicate	Rewritten
Overriding	$E \mapsto F \in f \lhd\!\!+ g$	$E \mapsto F \in (\mathrm{dom}\,(g) \lhd f) \ \cup \ g$
Direct product	$E \mapsto (F \mapsto G) \in f \otimes g$	$E \mapsto F \in f \ \ \wedge \ \ E \mapsto G \in g$
Parallel product	$(E \mapsto F) \mapsto (G \mapsto H) \in f \parallel g$	$E \mapsto G \in f \ \ \wedge \ \ F \mapsto H \in g$

The overriding operator is applicable in general to a relation f from, say, a set S to a set T, and a relation g also from S to T. Figure 5 illustrates overriding.

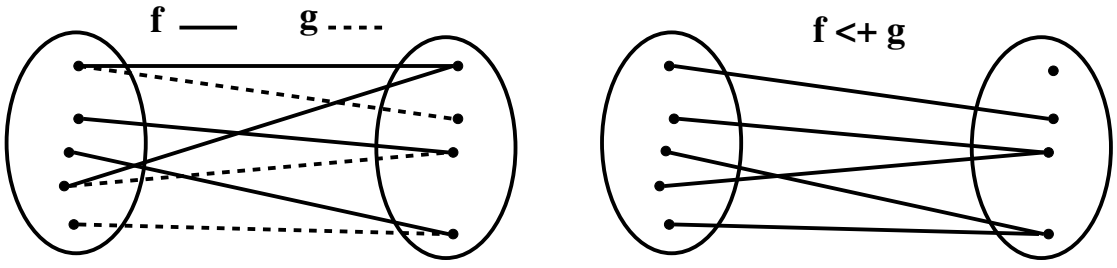


Fig. 5. Relation Overriding

When f is a function and g is the singleton function $\{x \mapsto E\}$, then $f \leftarrow \{x \mapsto E\}$ replaces in f the pair $x \mapsto f(x)$ by the pair $x \mapsto E$. Notice that in the case where x is not in the domain of f, then $f \leftarrow \{x \mapsto E\}$ simply adds the pair $x \mapsto E$ to the function f. In this case, it is thus equal to $f \cup \{x \mapsto E\}$.

5.6 Functional Operators

In this section we define various function operators: the sets of all partial and total functions, partial and total injections, partial and total surjections, and bijections. We also introduce the two projection functions as well as the identity function.

$$
\begin{array}{rl}
expression ::= & \ldots \\
& \mathrm{id} \\
& expression \pfun expression \\
& expression \fun expression \\
& expression \pinj expression \\
& expression \inj expression \\
& expression \psur expression \\
& expression \surj expression \\
& expression \bij expression \\
& \mathrm{prj}_1 \\
& \mathrm{prj}_2
\end{array}
$$

Operator	Predicate	Rewritten
Identity	$E \mapsto F \in \mathrm{id}$	$E = F$
Set of all partial functions	$f \in S \pfun T$	$f \in S \rel T \ \land \ (f^{-1} ; f) \subseteq \mathrm{id}$
Set of all total functions	$f \in S \fun T$	$f \in S \pfun T \ \land \ S = \mathrm{dom}(f)$
Set of all partial injections	$f \in S \pinj T$	$f \in S \pfun T \ \land \ f^{-1} \in T \pfun S$
Set of all total injections	$f \in S \inj T$	$f \in S \fun T \ \land \ f^{-1} \in T \pfun S$
Set of all partial surjections	$f \in S \psur T$	$f \in S \pfun T \ \land \ T = \mathrm{ran}(f)$
Set of all total surjections	$f \in S \surj T$	$f \in S \fun T \ \land \ T = \mathrm{ran}(f)$
Set of all bijections	$f \in S \bij T$	$f \in S \inj T \ \land \ f \in S \surj T$

Operator	Predicate	Rewritten
First projection	$(E \mapsto F) \mapsto G \in \mathrm{prj}_1$	$G = E$
Second projection	$(E \mapsto F) \mapsto G \in \mathrm{prj}_2$	$G = F$

5.7 Summary of the Arrows

Operator	Arrow
binary relations	$S \leftrightarrow T$
total relations	$S \nleftrightarrow T$
surjective relations	$S \twoheadleftarrow T$
total surjective relations	$S \nleftrightarrow T$
partial functions	$S \nrightarrow T$
total functions	$S \rightarrow T$

Operator	Arrow
partial injections	$S \rightarrowtail T$
total injections	$S \rightarrowtail T$
partial surjections	$S \twoheadrightarrow T$
total surjections	$S \twoheadrightarrow T$
bijections	$S \rightarrowtail\!\!\!\!\twoheadrightarrow T$

5.8 Lambda Abstraction and Function Invocation

We now define *lambda abstraction*, which is a way to construct functions, and also function invocation, which is a way to call functions. But first we have to define the notion of *pattern of variables*. A pattern of variables is either an identifier or a pair made of two patterns of variables. Moreover, all variables composing the pattern must be distinct. For example, here are three patterns of variables:

$$\mathsf{abc}$$

$$\mathsf{abc} \mapsto \mathsf{def}$$

$$\mathsf{abc} \mapsto (\mathsf{def} \mapsto \mathsf{ghi})$$

Given a pattern of variables x, a predicate P, and an expression E, the construct $\lambda\, x \cdot P \mid E$ is a lambda abstraction, which is a function. Given a function f and an expression E, the construct $f(E)$ is an expression denoting a function invocation. Here is our new syntax:

$$
\begin{array}{lll}
expression & ::= & \ldots \\
 & & expression(expression) \\
 & & \lambda\, pattern \cdot predicate \mid expression \\
 & & \\
pattern & ::= & variable \\
 & & pattern \mapsto pattern
\end{array}
$$

In the following table, l stands for the list of variables in the pattern L.

Operator	Predicate	Rewritten
Lambda abstraction	$F \in \lambda L \cdot P \mid E$	$F \in \{l \cdot P \mid L \mapsto E\}$
Function invocation	$F = f(E)$	$E \mapsto F \in f$

The function invocation construct $f(E)$ requires a well-definedness condition, which is the following:

Expression	Well-definedness condition
$f(E)$	$f^{-1} ; f \subseteq \mathrm{id} \quad \wedge \quad E \in \mathrm{dom}(f)$

6 Boolean and Arithmetic Language

6.1 Syntax

In this section, we extend once more the expressions. An expression might be a boolean or a number. Booleans are either TRUE or FALSE (do not confuse them with $\top$ and $\bot$). Numbers are either 0, 1, ..., the sum, product, or power of two numbers. We also add the sets BOOL, $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{N}1$ and the functions succ and pred.

$$
\begin{array}{rl}
expression ::= & \ldots \\
& \text{BOOL} \\
& \text{TRUE} \\
& \text{FALSE} \\
& \mathbb{Z} \\
& \mathbb{N} \\
& \mathbb{N}_1 \\
& \text{succ} \\
& \text{pred} \\
& 0 \\
& 1 \\
& \ldots \\
& expression + expression \\
& expression * expression \\
& expression \mathbin{\hat{}} expression
\end{array}
$$

6.2 Peano Axioms and Recursive Definitions

The following predicates yield definition of the boolean and arithmetic expressions:

$$\text{BOOL} \;=\; \{\text{TRUE}, \text{FALSE}\}$$

$$\text{TRUE} \neq \text{FALSE}$$

$$0 \;\in\; \mathbb{N}$$

$$\text{succ} \;\in\; \mathbb{Z} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{Z}$$

$$\text{pred} \;=\; \text{succ}^{-1}$$

$$\forall\, S \cdot 0 \in S \;\;\wedge\;\; (\forall\, n \cdot n \in S \;\Rightarrow\; \text{succ}(n) \in S) \;\;\Rightarrow\;\; \mathbb{N} \subseteq S$$

$$\forall\, a \cdot a + 0 \;=\; a$$

$$\forall\, a \cdot a * 0 \;=\; 0$$

$$\forall\, a \cdot a \mathbin{\hat{}} 0 \;=\; \text{succ}(0)$$

$$\forall\, a, b \cdot a + \text{succ}(b) = \text{succ}(a + b)$$

$$\forall\, a, b \cdot a * \text{succ}(b) = (a * b) + a$$

$$\forall\, a, b \cdot a \mathbin{\hat{}} \text{succ}(b) = (a \mathbin{\hat{}} b) * a$$

6.3 Extension of the Arithmetic Language

We introduce the classical binary relations on numbers, the finiteness predicate, the interval between two numbers, the subtraction, division, modulo, cardinal, maximum, and minimum constructs.

$$
\begin{array}{lll}
\ldots & & \\
predicate & ::= & \ldots \\
 & & expression \leq expression \\
 & & expression < expression \\
 & & expression \geq expression \\
 & & expression > expression \\
 & & \text{finite}(expression) \\
 & & \\
expression & ::= & \ldots \\
 & & expression \mathinner{..} expression \\
 & & expression - expression \\
 & & expression \,/\, expression \\
 & & expression \ \text{mod}\ expression \\
 & & \text{card}(expression) \\
 & & \max(expression) \\
 & & \min(expression)
\end{array}
$$

Operator	Predicate	Rewritten
smaller than or equal	$a \leq b$	$\exists c \cdot c \in \mathbb{N} \ \wedge \ b = a + c$
smaller than	$a < b$	$a \leq b \ \wedge \ a \neq b$
greater than or equal	$a \geq b$	$\neg\,(a < b)$
greater than	$a > b$	$\neg\,(a \leq b)$
interval	$c \ \in \ a \mathinner{..} b$	$a \leq c \ \wedge \ c \leq b$
subtraction	$c = a - b$	$a = b + c$
division	$c = a/b$	$\exists r \cdot (\, r \in \mathbb{N} \ \wedge \ r < b \ \wedge \ a = c * b + r \,)$
modulo	$r = a \ \text{mod}\ b$	$a = (a/b) * b + r$
finiteness	$\text{finite}(s)$	$\exists n, f \cdot n \in \mathbb{N} \ \wedge \ f \in 1 \mathinner{..} n \rightarrowtail\!\!\!\!\!\rightarrow s$
cardinality	$n = \text{card}(s)$	$\exists f \cdot f \ \in \ 1 \mathinner{..} n \rightarrowtail s$

Operator	Predicate	Rewritten
maximum	$n = \max(s)$	$n \in s \ \wedge \ (\forall x \cdot x \in s \ \Rightarrow x \leq n)$
minimum	$n = \min(s)$	$n \in s \ \wedge \ (\forall x \cdot x \in s \ \Rightarrow x \geq n)$

Division, modulo, cardinal, minimum, and maximum are subjected to some well-definedness conditions, which are the following:

Numeric Expression	Well-definedness condition
a/b	$b \neq 0$
$a \mod b$	$0 \leq a \ \wedge \ b > 0$
$\mathrm{card}(s)$	$\mathrm{finite}(s)$
$\max(s)$	$s \neq \varnothing \ \wedge \ \exists x \cdot (\forall n \cdot n \in s \ \Rightarrow \ x \geq n)$
$\min(s)$	$s \neq \varnothing \ \wedge \ \exists x \cdot (\forall n \cdot n \in s \ \Rightarrow \ x \leq n)$

7 Advanced Data Structures

In this section, we show how our basic mathematical language can still be extended to cope with some classical (advanced) data structures we shall use in subsequent chapters of the book, essentially strongly connected graphs, lists, rings and trees. We present the axiomatic definitions of these data structures together with some theorems. We do not present proofs. In fact all such proofs have been done with the Rodin Platform.

7.1 Irreflexive Transitive Closure

We start with the definition of the irreflexive transitive closure of a relation, which is a very useful concept to be used in what follows. Given a relation r from a set S to itself, the irreflexive transitive closure of r, denoted by cl(r), is also a relation from S to S. The characteristic properties of cl(r) are the following:

1. Relation r is included in cl(r)
2. The forward composition of cl(r) with r is included in cl(r)

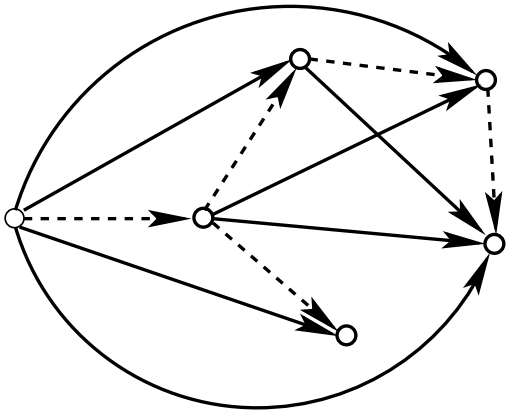


Fig. 6. A Relation (dashed) and its Irreflexive Transitive Closure (dashed and plain)

3. Relation cl(r) is the smallest relation dealing with 1 and 2

This is illustrated in figure 8. It can be formalized as follows:

$$\begin{array}{ll}
\mathbf{axm_1}: & r \in S \leftrightarrow S \\
\mathbf{axm_2}: & \mathrm{cl}(r) \in S \leftrightarrow S \\
\mathbf{axm_3}: & r \subseteq \mathrm{cl}(r) \\
\mathbf{axm_4}: & \mathrm{cl}(r) ; r \subseteq \mathrm{cl}(r) \\
\mathbf{axm_5}: & \forall p \cdot r \subseteq p \ \wedge \ p ; r \subseteq p \ \Rightarrow \ \mathrm{cl}(r) \subseteq p
\end{array}$$

The following theorems can be proved:

$$\begin{array}{ll}
\mathbf{thm_1}: & \mathrm{cl}(r) ; \mathrm{cl}(r) \subseteq \mathrm{cl}(r) \\
\mathbf{thm_2}: & \mathrm{cl}(r) = r \cup r ; \mathrm{cl}(r) \\
\mathbf{thm_3}: & \mathrm{cl}(r) = r \cup \mathrm{cl}(r) ; r \\
\mathbf{thm_4}: & \forall s \cdot r[s] \subseteq s \ \Rightarrow \ \mathrm{cl}(r)[s] \subseteq s \\
\mathbf{thm_5}: & \mathrm{cl}(r^{-1}) = \mathrm{cl}(r)^{-1}
\end{array}$$

These theorems are proved by finding some instantiations for the local variable p in the universally quantified axiom **axm_5**. In particular, the proof of **thm_1** is handled by instantiating p with [1]:

$$\{ x \mapsto y \mid cl(r) ; \{x \mapsto y\} \subseteq cl(r) \}$$

[1] This was suggested by D. Cansell

7.2 Strongly Connected Graphs

Given a set V and a binary relation r from V to itself, the graph representing this relation is said to be *strongly connected* if any two distinct points m and n in V are possibly connected by a path built on r. This is illustrated on figure 7. This can be formalized as follows:

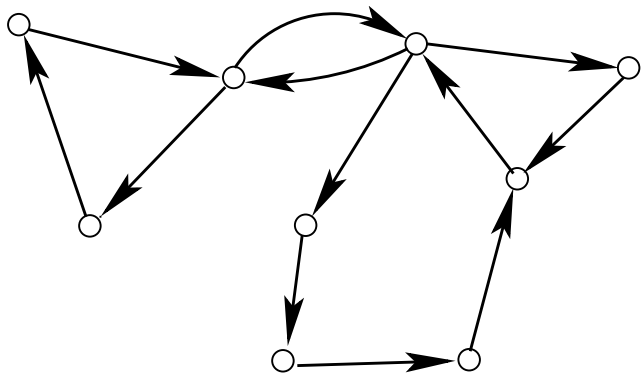


Fig. 7. A Strongly Connected Graph

$$\begin{array}{ll} \mathbf{axm\_1}: & r \in V \leftrightarrow V \\ \mathbf{axm\_2}: & (V \times V) \setminus \mathrm{id} \subseteq \mathrm{cl}(r) \end{array}$$

This definition is easy to understand: it simply says that every two distinct points of V are related through the irreflexive transitive closure cl(r) of r. But this definition is not very convenient to use in proof. Here is an *equivalent one* which is more convenient:

$$\mathbf{thm\_1}: \quad \forall S \cdot S \neq \varnothing \ \wedge \ r[S] \subseteq S \ \Rightarrow \ V \subseteq S$$

The intuition behind this definition is the following: it says that the only set S (except the empty set) which is such that $r[S] \subseteq S$ is the entire set V. For example, suppose we have

$$V = \{a, b\}$$

$$r = \{a \mapsto b\}$$

$$r[\{a\}] = \{b\}$$

$$r[\{b\}] = \varnothing$$

$$r[\{a, b\}] = \{b\}$$

The graph r is not connected because the non-empty set $\{b\}$, which is different from V, is such that $r[\{b\}] \subseteq \{b\}$. Now suppose:

$$V = \{a, b\}$$

$$r = \{a \mapsto b, b \mapsto a\}$$

$$r[\{a\}] = \{b\}$$

$$r[\{b\}] = \{a\}$$

$$r[\{a, b\}] = \{a, b\}$$

The graph r is strongly connected since the only non-empty set S where $r[S] \subseteq S$ is $\{a, b\}$, that is indeed V.

Also note the following result which is very intuitive: if r is strongly connected then so is r^{-1}.

7.3 Infinite Lists

An infinite list built on a set V is defined by means of a point f of V (the beginning of the list) and a bijective function n from V to $V \setminus \{f\}$. It is illustrated on figure 8.

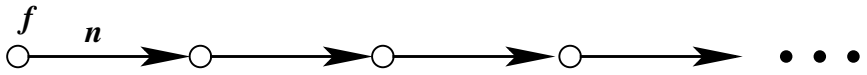


Fig. 8. An Infinite List

This can be formalized as follows:

$$\begin{array}{ll} \textbf{axm\_1}: & f \in V \\ \textbf{axm\_2}: & n \in V \rightarrowtail\!\!\!\!\!\rightarrow V \setminus \{f\} \end{array}$$

But these two properties are not enough. We need a final property which says that there are no cycles or backward infinite chains, which are not precluded by axioms **axm_1** and **axm_2**. We want to eliminate the backward infinite chain and the cycle which are shown on figure 9.

A set S containing a cycle or an infinite backward chain is such that each point x in S is related to a point y in S by the relation n^{-1}. This can be formalized as follows:

$$\forall x \cdot x \in S \;\Rightarrow\; (\,\exists y \cdot y \in S \;\wedge\; y \mapsto x \in n\,)$$

that is:

$$S \;\subseteq\; n[S]$$

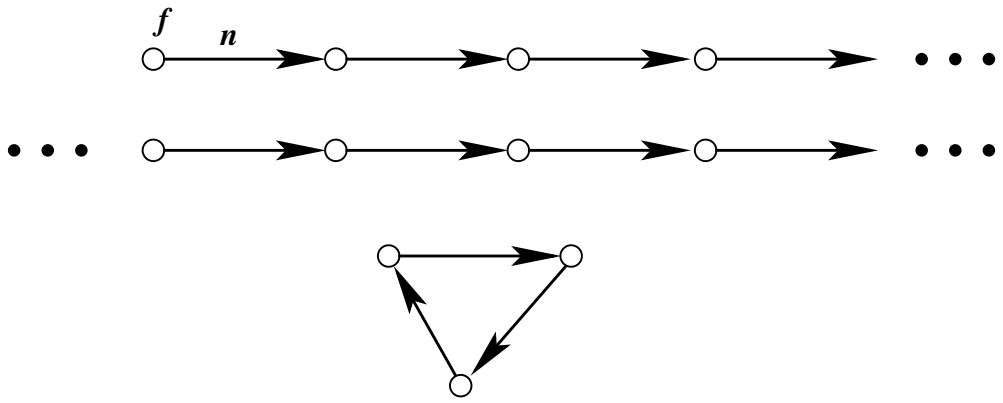


Fig. 9. Avoiding Infinite Backward Chains and Cycles

But as the empty set enjoys this property, we state in the following axiom that the only set with that property is precisely the empty set:

$$\mathbf{axm\_3}: \quad \forall S \cdot S \subseteq n[S] \;\Rightarrow\; S = \varnothing$$

A classical example of an infinite list is one where V is the set of natural number $\mathbb{N}$, f is 0, and n is the successor function succ restricted to $\mathbb{N}$. This is illustrated on figure 10.

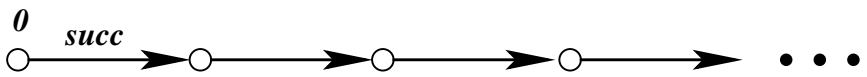


Fig. 10. The Natural Numbers

It clearly obeys **axm_1** and **axm_2**. It also obeys **axm_3**: let S be a non-empty subset of $\mathbb{N}$ then $\mathrm{succ}[S]$ does not contain $\min(S)$ thus S is not included in $\mathrm{succ}[S]$. In fact, **axm_1** and **axm_2** are exactly the first four Peano axioms. But clearly **axm_3** does not correspond to the last Peano axiom (recurrence). However, the following theorem shows that the last Peano axiom can be proved from **axm_3** (and vice-versa). This can be done easily by instantiating S in **axm_3** with $V \setminus T$:

$$\mathbf{thm\_1}: \quad \forall T \cdot f \in T \;\wedge\; n[T] \subseteq T \;\Rightarrow\; V \subseteq T$$

By unfolding $n[T] \subseteq T$, we obtain:

$$\mathbf{thm\_2}: \quad \forall T \cdot f \in T \;\wedge\; (\forall x \cdot x \in T \Rightarrow n(x) \in T) \;\Rightarrow\; V \subseteq T$$

Translating this to the natural numbers, we obtain the last Peano axiom

$$\forall T \cdot 0 \in T \ \wedge \ (\forall x \cdot x \in T \Rightarrow x + 1 \in T) \ \Rightarrow \ \mathbb{N} \subseteq T$$

Next are three more theorems which might be useful. Observe **thm_4** which says that backward chaining is finite. Theorem **thm_5** represents another way to state that there are no cycles. It does not say however that there is no backward infinite chains, so it is not equivalent to **axm_3**: it is only implied by **axm_3**.

$$\begin{array}{ll} \mathbf{thm\_3}: & \mathrm{cl}(n)[\{f\}] \cup \{f\} = V \\ \mathbf{thm\_4}: & \forall x \cdot \ \mathrm{finite}(cl(n^{-1})[\{x\}]) \\ \mathbf{thm\_5}: & cl(n) \cap \mathrm{id} = \varnothing \end{array}$$

Note that in the case of the natural numbers, $a \mapsto b \in \mathrm{cl}(\mathrm{succ})$ is the same as $a < b$, and $\mathrm{cl}(\mathrm{succ}^{-1})[\{a\}] \cup \{a\}$ (for any natural number a) is the same as the interval $0 \mathinner{..} a$.

The List Induction Rule. Theorem **thm_2** can be used to prove a property $\mathsf{P}(x)$ for all nodes of a list. It is done in the following fashion. The property $\mathsf{P}(x)$ is transformed into the following set:

$$\{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \}$$

And now proving that $\mathsf{P}(x)$ holds for each node x of V is exactly the same as proving that V is included into that set, that is:

$$V \ \subseteq \{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \}$$

For doing so, it suffices to instantiate T in **thm_2** with the set $\{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \}$. This yields:

$$\begin{array}{l} f \ \in \ \{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \} \\ \forall x \cdot x \in \{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \} \ \Rightarrow \ n(x) \in \{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \} \\ \Rightarrow \\ V \ \subseteq \{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \} \end{array}$$

The first antecedent of this implication reduces to:

$$\mathsf{P}(f)$$

The second antecedent can be rewritten:

$$\forall x \cdot x \in V \ \wedge \ \mathsf{P}(x) \ \Rightarrow \ \mathsf{P}(n(x))$$

And now, once we have proved the previous statements then we can deduce the following which was our initial goal:

$$V \subseteq \{ x \mid x \in V \ \wedge \ \mathsf{P}(x) \}$$

that is

$$\forall x \cdot x \in V \ \Rightarrow \ \mathsf{P}(x)$$

To summarize, when one has to prove a property $\mathsf{P}(x)$ for all elements x of a list, a possibility is to do the following:

- prove that $\mathsf{P}(f)$ holds for the first element f of the list,
- prove that $\mathsf{P}(n(x))$ holds for any x in V, under the assumption that $\mathsf{P}(x)$ holds.

In doing so, the property $\mathsf{P}(x)$ is said to be proved *by list-induction*. All this can now be transformed in an inference rule as follows:

$$\frac{\mathsf{H} \vdash \mathsf{P}(f) \qquad \mathsf{H},\ x \in V,\ \mathsf{P}(x) \vdash \mathsf{P}(n(x))}{\mathsf{H},\ x \in V \vdash \mathsf{P}(x)} \quad \begin{array}{c}\mathsf{IND\_LIST} \\ (\mathbf{x}\ \underline{\mathsf{nfin}}\ \mathsf{H})\end{array}$$

By translating this rule to the Natural Number, this yields:

$$\frac{\mathsf{H} \vdash \mathsf{P}(0) \qquad \mathsf{H},\ x \in \mathbb{N},\ \mathsf{P}(x) \vdash \mathsf{P}(x+1)}{\mathsf{H},\ x \in \mathbb{N} \vdash \mathsf{P}(x)} \quad \begin{array}{c}\mathsf{IND\_}\mathbb{N} \\ (\mathbf{x}\ \underline{\mathsf{nfin}}\ \mathsf{H})\end{array}$$

7.4 Finite Lists

A finite list constructed on the set V is defined by means of two points f (denoting the first element in the list) and l (denoting the last element in the list). The list itself is a bijection. It is illustrated on figure 11. Finally, an axiom similar to axiom **axm_3** of the infinite lists says that there is no backward chain nor cycles.

$$\begin{array}{ll} \mathbf{axm\_1}: & f \in V \\ \mathbf{axm\_2}: & l \in V \\ \mathbf{axm\_3}: & n \in V \setminus \{l\} \rightarrowtail\!\!\!\!\!\rightarrow V \setminus \{f\} \\ \mathbf{axm\_4}: & \forall S \cdot S \subseteq n[S] \ \Rightarrow \ S = \varnothing \end{array}$$

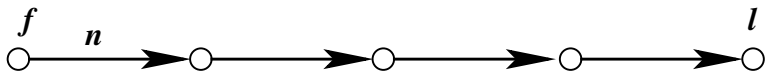


Fig. 11. A Finite List

Notice that axiom **axm_4** is not symmetric with regard to both directions on the list. But this can be proved in a systematic manner. This is what is shown in the following theorems:

thm_1 :	$\forall T \cdot f \in T \ \wedge \ n[T] \subseteq T \ \Rightarrow \ V \subseteq T$
thm_2 :	$\text{cl}(n)[\{f\}] \cup \{f\} = V$
thm_3 :	$\text{cl}(n^{-1})[\{l\}] \cup \{l\} = V$
thm_4 :	$\forall T \cdot l \in T \ \wedge \ n^{-1}[T] \subseteq T \ \Rightarrow \ V \subseteq T$
thm_5 :	$\forall S \cdot S \subseteq n^{-1}[S] \ \Rightarrow \ S = \varnothing$
thm_6 :	$\text{finite}(V)$
thm_7 :	$cl(n) \cap \text{id} = \varnothing$

A classical example of finite lists are numerical intervals $a \mathinner{..} b$ (with $a \leq b$). This is illustrated on figure 12.

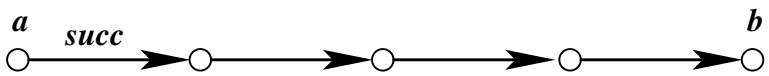


Fig. 12. A Numerical Interval

It is easy to prove the following:

$$a \in a \mathinner{..} b$$

$$b \in a \mathinner{..} b$$

$$(a \mathinner{..} b - 1) \lhd \text{succ} \ \in \ (a \mathinner{..} b) \setminus \{b\} \rightarrowtail\!\!\!\!\!\rightarrow (a \mathinner{..} b) \setminus \{a\}$$

Coming back to our general finite lists, let us now define the set of elements itvl(x) belonging to a sublist from f to x on a finite list from f to l:

axm_5 :	$\text{itvl} \in V \rightarrow \mathbb{P}(V)$
axm_6 :	$\forall x \cdot x \in V \ \Rightarrow \ \text{itvl}(x) = cl(n^{-1})[\{x\}] \cup \{x\}$

The following theorems state some useful properties of these sets. Observe the recursive property stated in **thm_9**.

$$\begin{array}{ll}
\mathbf{thm_8}: & \forall x \cdot x \in V \ \Rightarrow\ \{f, x\} \subseteq \text{itvl}(x) \\
\mathbf{thm_9}: & \forall x \cdot x \in V \setminus \{f\} \ \Rightarrow\ \text{itvl}(x) = \text{itvl}(n^{-1}(x)) \cup \{x\} \\
\mathbf{thm_10}: & \text{itvl}(l) = V
\end{array}$$

The last theorem is just a rewording of **thm_3**.

7.5 Rings

A ring is defined by a bijection which is strongly connected. It is illustrated on figure 13. Thus we copy in **axm_2** part of the statement of **thm_2** of section 7.2 showing the equivalence to strong connectivity.

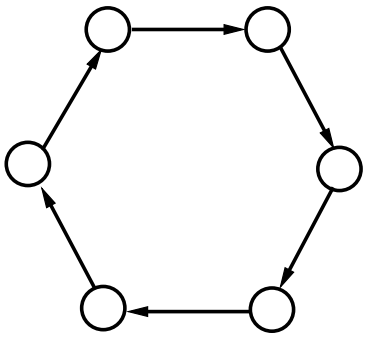


Fig. 13. A Ring

$$\begin{array}{ll}
\mathbf{axm_1}: & n \in V \rightarrowtail V \\
\mathbf{axm_2}: & \forall S \cdot S \neq \varnothing \ \wedge n^{-1}[S] \subseteq S \ \Rightarrow\ V \subseteq S
\end{array}$$

Since n is injective (bijective, in fact), we have the following:

$$\mathbf{thm\_1}: \quad \forall S \cdot n^{-1}[S] \subseteq S \ \Leftrightarrow\ S \subseteq n[S]$$

This allows us to transform as follows the connectivity relationship of the ring:

$$\textbf{thm\_2}: \quad \forall S \cdot S \neq \varnothing \ \wedge \ S \subseteq n[S] \ \Rightarrow \ V \subseteq S$$

By cutting a ring between $n^{-1}(x)$ and x, we obtain a finite list from x to $n^{-1}(x)$. This is illustrated on figure 14. This finite list starts at x and ends at $n^{-1}(x)$. This is stated in the following theorems:

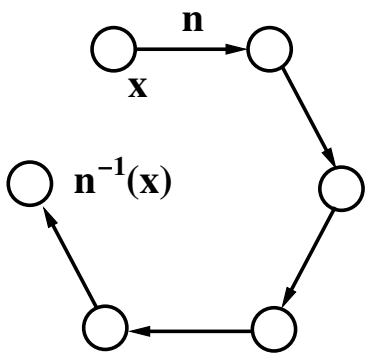


Fig. 14. A Cut Ring

$$\textbf{thm\_1}: \quad \forall x \cdot x \in V \ \Rightarrow \ p \in V \setminus \{n^{-1}(x)\} \rightarrowtail\!\!\!\!\!\rightarrow V \setminus \{x\}$$

$$\textbf{thm\_2}: \quad \forall x \cdot x \in V \ \Rightarrow \ (\forall S \cdot S \subseteq p[S] \ \Rightarrow \ S = \varnothing)$$

where p is $n \rhd \{x\}$

Let us now define the sets of elements itvr$(x)(y)$ belonging to an interval on a ring from x to y.

$$\textbf{axm\_3}: \quad \text{itvr} \in V \rightarrow (V \rightarrow \mathbb{P}(V))$$

$$\textbf{axm\_4}: \quad \forall x, y \cdot x \in V \wedge y \in V \ \Rightarrow \ \text{itvr}(x)(y) = cl(\{x\} \lhd n^{-1})[\{y\}] \cup \{y\}$$

The following theorems state some useful properties of the intervals:

$$\textbf{thm\_3}: \quad \forall x \cdot x \in V \wedge y \in V \ \Rightarrow \ \{x, y\} \subseteq \text{itvr}(x)(y)$$

$$\textbf{thm\_4}: \quad \forall x \cdot x \in V \wedge y \in V \setminus \{x\} \ \Rightarrow \ \text{itvr}(x)(y) = \text{itvr}(x)(n^{-1}(y)) \cup \{y\}$$

$$\textbf{thm\_5}: \quad \forall x \cdot x \in V \ \Rightarrow \ \text{itvr}(x)(n^{-1}(x)) = V$$

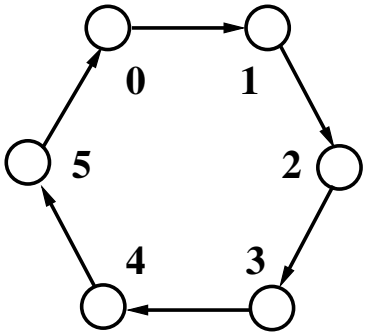


Fig. 15. A Ring Modulo 6

The last theorem is an adaptation of theorem **thm_10** of finite lists. A classical example of a ring is given by the "addition-modulo" as illustrated on figure 15.

Notice that sometimes it is more convenient to use a ring than an "addition-modulo": proofs are getting simpler.

7.6 Infinite Trees

Infinite trees generalise infinite lists. The beginning f of the list is replaced by the top t of the tree. The function p replaces n^{-1} of the infinite list. This is illustrated on figure 18. This is expressed by axioms **axm_1** and **axm_2** below. Axiom **axm_3** has the same function as **axm_3** on infinite lists: it removes cycles and infinite backward chains:

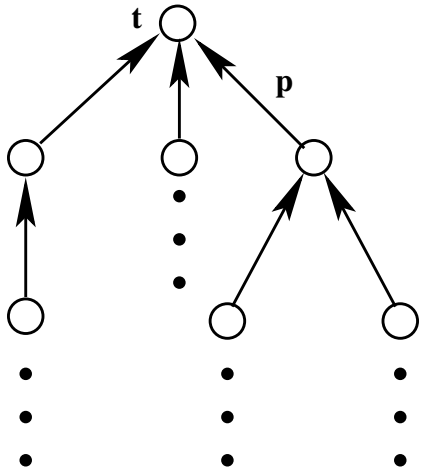


Fig. 16. An Infinite Tree

$$\begin{array}{ll} \textbf{axm\_1}: & t \in V \\ \textbf{axm\_2}: & p \in V \setminus \{t\} \twoheadrightarrow V \\ \textbf{axm\_3}: & \forall S \cdot S \subseteq p^{-1}[S] \;\Rightarrow\; S = \varnothing \end{array}$$

The next theorem defines an induction rules which generalise that of infinite lists.

$$\begin{array}{ll} \textbf{thm\_1}: & \forall T \cdot t \in T \;\wedge\; p^{-1}[T] \subseteq T \;\Rightarrow\; V \subseteq T \\ \textbf{thm\_2}: & \mathrm{cl}(p^{-1})[\{t\}] \cup \{t\} = V \end{array}$$

The following theorem states that backwards chains are finite.

$$\textbf{thm\_3}: \quad \forall x \cdot \mathrm{finite}(\mathrm{cl}(p)[\{x\}])$$

The List Induction Rule. It is easy to prove that **thm_1** is equivalent to the following theorem **thm_4** (hint: instantiate T in **thm_1** with $N \setminus T$):

$$\begin{array}{lll} \textbf{thm\_4:} & \forall T \cdot & T \subseteq V \\ & & t \in T \\ & & p^{-1}[T] \subseteq T \\ & & \Rightarrow \\ & & V \subseteq T \end{array}$$

This theorem can be further unfolded to the following equivalent one:

$$\begin{array}{lll} \textbf{thm\_5:} & \forall T \cdot & T \subseteq V \\ & & t \in T \\ & & \forall x \cdot x \in V \setminus \{t\} \;\wedge\; p(x) \in T \;\Rightarrow\; x \in T \\ & & \Rightarrow \\ & & V \subseteq T \end{array}$$

This is so because we have the following:

$$\begin{array}{l} p^{-1}[T] \;\subseteq\; T \\ \Leftrightarrow \\ \forall x \cdot x \in p^{-1}[T] \;\Rightarrow\; x \in T \\ \Leftrightarrow \\ \forall x \cdot (\,\exists y \cdot y \in T \;\wedge\; x \mapsto y \in p\,) \;\Rightarrow\; x \in T \\ \Leftrightarrow \\ \forall x \cdot (\,\exists y \cdot y \in T \;\wedge\; x \in \mathrm{dom}(p) \;\wedge\; y = p(x)\,) \;\Rightarrow\; x \in T \\ \Leftrightarrow \\ \forall x \cdot x \in V \setminus \{t\} \;\wedge\; p(x) \in T \;\Rightarrow\; x \in T \end{array}$$

Theorem **thm_5** can be used to prove a property $\mathsf{P}(x)$ for all nodes of a tree. It is done in the following fashion. The property $\mathsf{P}(x)$ is transformed into the following set:

$$\{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \}$$

And now proving that $\mathsf{P}(x)$ holds for each node x of V is exactly the same as proving that V is included into that set, that is:

$$V \ \subseteq \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \}$$

For doing so, it suffices to instantiate T in **thm_5** with the set $\{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \}$. This yields:

$$\begin{array}{l} \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \} \ \subseteq\ V \\ t \ \in\ \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \} \\ \forall x \cdot \left(\begin{array}{l} x \in V \setminus \{t\} \\ p(x) \in \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \} \\ \Rightarrow \\ x \in \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \} \end{array} \right) \\ \Rightarrow \\ V \ \subseteq \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \} \end{array}$$

The first antecedent of this implication is obvious because the set $\{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \}$ is indeed included in the set V, and the second antecedent reduces to:

$$\boxed{\mathsf{P}(t)}$$

The third antecedent can be rewritten:

$$\boxed{\forall x \cdot x \in V \setminus \{t\} \ \wedge\ \mathsf{P}(p(x)) \ \Rightarrow\ \mathsf{P}(x)}$$

And now, once we have proved the previous statements then we can deduce the following which was our initial goal:

$$V \ \subseteq \{\, x \mid x \in V \ \wedge\ \mathsf{P}(x) \}$$

that is

$$\forall x \cdot (\, x \in V \ \Rightarrow\ \mathsf{P}(x) \,)$$

To summarize, when one has to prove a property $\mathsf{P}(x)$ for all elements x of a tree, a possibility is to do the following:

– prove that $\mathsf{P}(t)$ holds for the top t of the tree,

– prove that $\mathsf{P}(x)$ holds for any x in $V \setminus \{t\}$, under the assumption that $\mathsf{P}(p(x))$ holds for the parent $p(x)$ of x in the tree.

In doing so, the property $\mathsf{P}(x)$ is said to be proved *by tree-induction*. All this can now be transformed in an inference rule as follows:

$$\frac{\mathsf{H} \vdash \mathsf{P}(t) \qquad \mathsf{H},\ x \in V \setminus \{t\},\ \mathsf{P}(p(x)) \vdash \mathsf{P}(x)}{\mathsf{H},\ x \in V \vdash \mathsf{P}(x)}\ \begin{array}{c}\mathsf{IND\_TREE}\\ (\mathbf{x}\ \underline{\mathsf{nfin}}\ \mathsf{H})\end{array}$$

7.7 Finite Depth Trees

Finite depth trees generalise finite lists. We still have a top point t which was f in the lists. But the last element l of the list is now replaced by a set L: these are the so-called leafs of the tree. All this is illustrated on figure 19. The axioms are as usual the following:

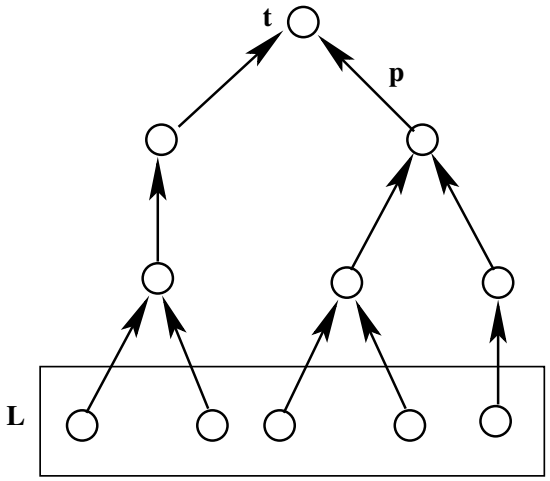


Fig. 17. A Finite Depth Tree

$$\begin{array}{ll}
\mathbf{axm_1}: & t \in V \\
\mathbf{axm_2}: & L \subseteq V \\
\mathbf{axm_3}: & p \in V \setminus \{t\} \rightarrow\!\!\!\!\!\rightarrow V \setminus L \\
\mathbf{axm_4}: & \forall S \cdot S \subseteq p^{-1}[S] \ \Rightarrow\ S = \varnothing \\
\mathbf{axm_5}: & \forall S \cdot S \subseteq p[S] \ \Rightarrow\ S = \varnothing
\end{array}$$

The theorems of finite lists can be adapted to finite depth trees as follows:

$$\begin{array}{ll}
\mathbf{thm_1}: & \forall T \cdot t \in T \ \wedge \ p^{-1}[T] \subseteq T \ \Rightarrow \ V \subseteq T \\
\mathbf{thm_2}: & \forall T \cdot L \subseteq T \ \wedge \ p[T] \subseteq T \ \Rightarrow \ V \subseteq T \\
\mathbf{thm_3}: & \mathrm{cl}(p^{-1})[\{t\}] \cup \{t\} = V \\
\mathbf{thm_4}: & \mathrm{cl}(p)[L] \cup L = V \\
\mathbf{thm_5}: & \forall x \cdot \mathrm{finite}(\mathrm{cl}(p)[\{x\}])
\end{array}$$

7.8 Free Trees

A free tree is a data structure which is often encountered in network modelling. Figure 18 shows a free tree. Given a finite set V (**axm_1**), a free tree is graph g with the following properties: it is a relation from V to V (**axm_2**), it is symmetric (**axm_3**), irreflexive (**axm_4**), connected (**axm_5**), and acyclic (**axm_6**) in spite of the symmetry.

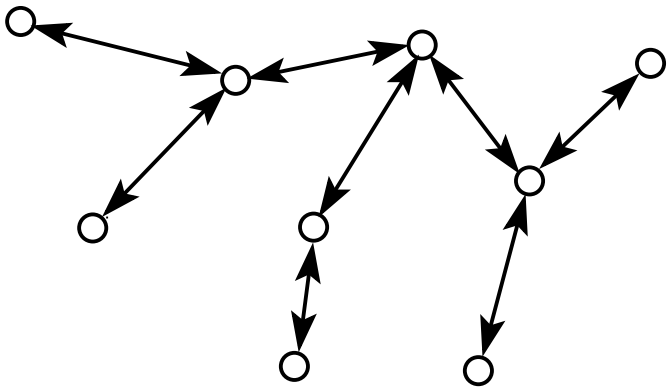


Fig. 18. A Free Tree

Axiom **axm_5** is a copy of theorem **thm_2** of section 7.2 dealing with strong connectivity. Note that axiom **axm_6** is not a copy of axiom **axm_4** of section 7.7: we added the quantified variable h and two properties, namely $h \cup h^{-1} = g$ and $h \cap h^{-1} = \varnothing$. This is due to the symmetry property of the graph, which one has somehow to "eliminate". The presence of h in **axm_6** has the effect of transforming the free tree into a finite tree. This is illustrated in figure 19.

$$
\begin{array}{ll}
\mathbf{axm\_1}: & \mathrm{finite}(V) \\
\mathbf{axm\_2}: & g \in V \leftrightarrow V \\
\mathbf{axm\_3}: & g \subseteq g^{-1} \\
\mathbf{axm\_4}: & g \cap \mathrm{id} = \varnothing \\
\mathbf{axm\_5}: & \forall S \cdot S \neq \varnothing \ \wedge \ g[S] \subseteq S \ \Rightarrow \ V \subseteq S \\
\mathbf{axm\_6}: & \forall h, S \cdot \ h \cup h^{-1} = g \\
 & \qquad\qquad h \cap h^{-1} = \varnothing \\
 & \qquad\qquad S \subseteq h[S] \\
 & \qquad\qquad \Rightarrow \\
 & \qquad\qquad S = \varnothing
\end{array}
$$

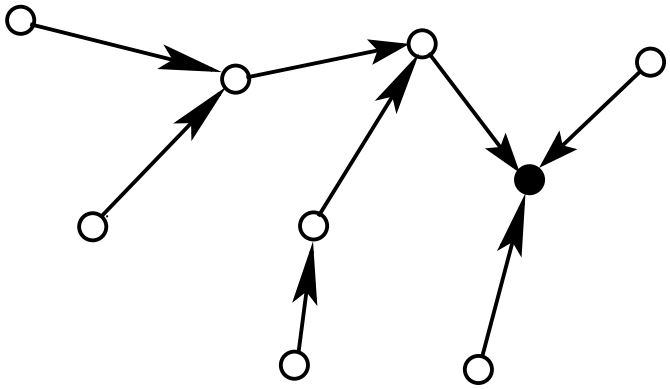


Fig. 19. A Free Tree Transformed into a Finite Tree

Outer and Inner Nodes of a Free Tree. The outer nodes of a free tree are those members x of the set V which are connected to a single node y in the free tree:

$$\{\, x \mid x \in V \ \wedge \ \exists y \cdot g[\{x\}] = \{y\} \,\}$$

The inner nodes are the other nodes. This is illustrated on figure 20 where the outer nodes are the black nodes while the inner nodes are the white ones. The following theorem states that when a free tree is not empty then its set of outer nodes is not empty either:

$$\mathbf{thm\_1}: \quad (\exists x \cdot V = \{x\}) \ \vee \ \exists x, y \cdot g[\{x\}] = \{y\}$$

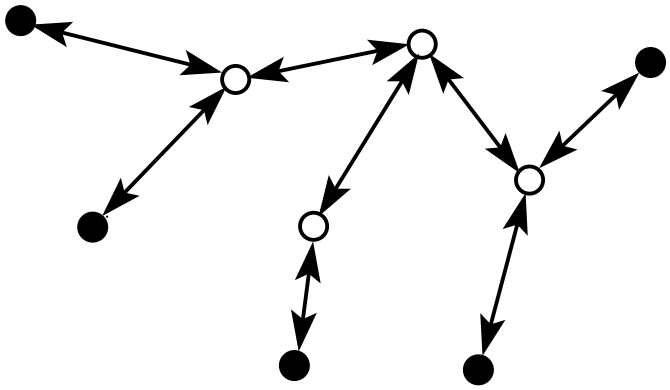


Fig. 20. The Outer and Inner Nodes of the Free Tree

7.9 Well-founded Relations and Directed Acyclic Graphs

We leave it to the reader to generalise infinite trees to well-founded graphs and finite trees to directed acyclic graphs.