A BACKWARDS THEOREM PROVER WITH FOCUSING, RESOURCE MANAGEMENT AND CONSTRAINTS FOR ROBOTIC PLANNING WITHIN INTUITIONISTIC LINEAR LOGIC

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Sıtar Kortik

January, 2010

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Uluç Saranlı (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Varol Akman

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda Nur Alpaslan

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

A BACKWARDS THEOREM PROVER WITH FOCUSING, RESOURCE MANAGEMENT AND CONSTRAINTS FOR ROBOTIC PLANNING WITHIN INTUITIONISTIC LINEAR LOGIC

Sıtar Kortik
M.S. in Computer Engineering
Supervisor: Asst. Prof. Dr. Uluç Saranlı
January, 2010

The main scope of this thesis is implementing a backwards theorem prover with focusing, resource management and constraints within the intuitionistic first-order linear logic for robotic planning problems. To this end, backwards formulations provide a simpler context for experimentation. However, existing backward theorem provers are either implemented without regard to the efficiency of the proof-search, or when they do, restrict the language to smaller fragments such as *Linear Hereditary Harrop Formulas* (*LHHF*). The former approach is unsuitable since it significantly impairs the scalability of the resulting system. The latter family of theorem provers address the scalability issue but impact the expressivity of the resulting language and may not be able to deal with certain non-deterministic planning elements. The proof theory we describe in this thesis enables us to effectively experiment with the use of linearity and continuous constraints to encode dynamic state elements characteristic of robotic planning problems. To this end, we describe a prototype implementation of our system in SWI-Prolog, and also incorporate continuous constraints into the prototype implementation of the system. We support the expressivity and efficiency of our system with some examples.

Keywords: constrained intuitionistic first-order linear logic, automated theorem proving, backwards theorem prover, robotic planning, SWI-Prolog implementation of *CFRM*.

ÖZET

ROBOTİK PLANLAMA İÇİN ODAKLANMA, KAYNAK YÖNETİMİ VE KISITLAMALARIN KATILARAK HEDEFE YÖNELİK TEOREM İSPATLAMANIN SEZGİSEL DOĞRUSAL MANTIKTA GERÇEKLEŞTİRİMİ

Sıtar Kortik
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Asst. Prof. Dr. Uluç Saranlı
Ocak, 2010

Bu tezin ana kapsamı, robotik planlama problemleri için, odaklanma, kaynak yönetimi ve kısıtlamaların da dahil edilerek, sezgisel doğrusal mantıkta, hedefe yönelik bir teorem ispatlama çatısı oluşturmak. Bu amaçla, hedefe yönelik formülasyon şekli, uygulama ve test aşamasında daha anlaşılır bir içerik sunmaktadır. Bununla beraber, mevcut hedefe yönelik teorem ispatlayıcılar, ispat aramada ya etkili bir yöntem sunamamaktadırlar ya da kullandıkları dili Doğrusal Hereditary Harrop Formülleri gibi daha küçük parçalara kısıtlayarak etkili bir yöntem sağlayabilmektedirler. Bahsedilen yaklaşımlardan ilki, sonuç sisteminin ölçeklenebilirliğine önemli derecede zarar verdiği için uygun değildir. İkinci bahsedilen teorem ispatlama yaklaşımlarında ise ölçeklenebilirlik konusu çözülebilir fakat ifade edebileceği dili kısıtlar ve belirli olmayan planlama elemanlarını ele alamayabilir. Bu tezde tanımladığımız ispatlama teorisi, robotik planlama problemlerindeki dinamik durum elemanlarının ifade edilmesinde, doğrusallığın ve sürekli kısıtlamaların etkili bir biçimde kullanılmasını sağlıyor. Bu amaçla, tanımladığımız sistemin SWI-Prolog dilinde bir uygulamasını gerçekleştirdik. Bu uygulamaya kısıtlamaları da dahil ederek sistemi genişlettik. Sistemimizin ifade gücünü ve verimliliğini, bazı robot planlama örnekleri vererek destekledik.

Anahtar sözcükler: kısıtlı sezgisel doğrusal mantık, özdevinimli kuram ispatlama, hedefe yönelik kuram ispatlama, robotik planlama, mantıksal çatı, *CFRM*'nin SWI-Prolog'da uygulaması.

Acknowledgement

I am heartily thankful to my advisor, Uluç Saranlı, whose encouragement, guidance and support from the initial to the final level enabled me to develop this thesis. Although I was new in the robotic area, his endless energy, knowdledge and patience helped me to like and be included in this area.

I also thank to my thesis committee, Varol Akman and Ferda Nur Alpaslan for their participation and giving advices for this thesis. I am also very grateful to Frank Pfenning for his collaboration and advices about our theorem prover system and also for his invaluable class notes of Constructive Logic, Automated Theorem Proving and Linear Logic lectures.

I am indebted to all members of *SensoRhex* Project group and Bilkent Dexterous Robotics and Locomotion (*BDRL*) group. Specially, I would like to thank Ömür Arslan, Tuğba Yıldız, Akın Avcı and Cihan Öztürk for not only our discussions about both robotics and casual conversations but also late night studyings.

Outside the research group, many people directly or indirectly contributed to my completion of this thesis. I thank to the Bilkent Aviation Club (*BILHAVK*), Bilkent Underwater Club (*BILSAT*) and all of their members for setting great social organizations. I am also appreciative to Computer Engineering Department of Bilkent University and European Commission for their financial support.

I owe my deepest gratitude to my parents, Hüseyin and Gönül Kortik, and my sister Rüçhan Kortik, for their endless love and support. Finally, I thank my beloved girlfriend Zeynep Çelen, for spicing my life, giving encouragement, motivating, understanding and supporting me through all years of living in different cities.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Scope

Day by day, robots are playing a bigger role in our lives. From cleaning robots in houses to Mars rovers like Phoenix [15], robotics is almost in all areas to facilitate people's lives. On earth, when a robot encounters a problem and is stuck while processing a task, people can intervene and help robot to solve that problem. This intervention is possible since we can communicate with robots easily. However for Mars rovers [1], this kind of intervention is almost impossible except sending some signals. Ideally, robots should achieve their tasks independently from people. This can be achieved with fully autonomous properties such as charging batteries, exploring new environments and reacting to changing conditions. However it is really hard to build a fully consistent system.

One of the biggest challenges in robotics is planning such as motion, task and etc. Robots have many internal parameters to manage such as joint torques or forces and the motion of the robot, and at the same time, reaction to changing environment is another challenging problem. There are many different approaches to planning for which we will now give some instances. Linear Temporal Logic (LTL) is used in [25] for artificial intelligence planning. Also in [12], motion planning using temporal logic for a mobile robot is presented. An automatic parking

system is also presented in [5] which shows that robot can automatically execute high level commands in changing environments. Other planning techniques reduce planning problem into a constraint satisfaction problem [7], and find plans by applying model checking techniques [33].

Logic based languages have always been attractive formalisms in which planning problems can be elegantly represented. However, their adoption in this context has been limited due to a number of important drawbacks. One major problem is that the computational complexity of reasoning systems based on theorem proving dramatically increases with their expressivity. While logic programming systems based on less expressive fragments such as Prolog, Lolli [18] etc. offer tractable alternatives, the range of planning problems that they can effectively encode are also limited.

An orthogonal but related issue is the compatibility of the logical formalism of choice with the problem domain of interest. In general, the concept of planning has an inherently dynamic nature, where relevant properties of an environment change as a result of actions taken by active agents. This domain also has a programmatic flavor since plans have a natural correspondence to possibly reactive program fragments. Along with various researchers in the field, we also believe that one of the best logical formalisms that can simultaneously address both of these issues is intuitionistic first-order linear logic. The choice of an intuitionistic formulation over a classical one is motivated by our desire to transform proofs in our system into executable behavioral plans. This would not be possible within classical logic, where the abstract nature of truth cannot always be associated with procedural proofs. Linearity is incorporated to deal with dynamic state elements for which a static notion of truth leads to very inefficient and practically infeasible encodings. Finally, a first-order language is necessary both to admit concise description of various robotic planning problems within the logic as well as supporting eventual integration with continuous constraints to model physical properties of systems being modeled.

1.2 Contributions

In this thesis, our main contribution is the specification and implementation an efficient backward sequent calculus for intuitionistic first-order linear logic with focusing and resource management (*FRM*). Our primary motivation is the use of intuitionistic linear logic for nontrivial robotic planning problems. We prefer the backward sequent calculus to the potentially more efficient forward search (i.e. the inverse method [4]) since the latter is significantly more difficult to implement. Moreover, we also prove that FRM system is sound and complete, for which details can be found in Appendix C. Our second major contribution is the integration of continuous constraints into the proof theory *FRM*. We call the resulting new system *CFRM*. Third, we incorporate annotated proof terms into *CFRM* system in order to support recording and extraction of plans corresponding to the constructed proofs. Finally, we implement this proof theory in SWI-Prolog and define a few small robotic planning domains to illustrate the utility of our contributions and the implementation.

1.3 Organization of the Thesis

In Chapter 2, we give some background on linear logic in general, followed by focused intuitionistic first-order linear logic (*FocLL*) [31] as well as some recent work on planning with logical languages. In Chapter 3, we introduce a novel automated theorem prover system, *FRM*, based on [3]. To eliminate non-determinism caused by disjunction and resource consumption, we incorporate focusing and resource management into this new system. The reader can also find the proof of soundness and completeness of the new system at the end of Chapter 3. In Chapter 4, we describe Constrained Intuitionistic Linear Logic (*CILL*) and give several examples. We also describe *CFRM*, a new theorem prover system with constraints incorporated. To be able to extract plans from *CFRM* proofs, we describe proof terms. Implementation details and experiment results are given in Chapter 5. The last chapter concludes the thesis and discusses future work.

Chapter 2

Background

2.1 Planning for Robotics

In this section, we give some background on planning and some examples related to planning in robotics. A *planner* is a special-purpose algorithm to search for a solution [37]. Many planners use the "classical" approach which describes states and operators in a restricted language, such as the STRIPS (STanford Research Institute Problem Solver) language [9].

As in the [37], we categorize planning as being done either in the space of situations or in the space of plans. *Situation space* planners search through the space of possible situations. We will describe two kinds of situation space planners and two kinds of plan space planners. The first situation space planner is the *situation calculus* and the second one is the *fluent calculus* which is an extension of the situation calculus. The first plan space planner is the *partial order planning* and the second one is the *total order planning*.

2.1.1 Situation Calculus

Situation calculus was first introduced by John McCarthy in 1963 [26, 30, 28]. It was designed to represent and reason about dynamical domains. The situation calculus has the three main elements: *Actions*, *situations* and *fluents*.

- *Actions* change the world which we define at the beginning. *POSS* is a special predicate which is used to indicate that an action is executable. The constant and function symbols for actions are completely dependent on the application.
- *Situations* are finite sequences of actions. A situation is not a state, it is a history of action occurrences. The initial situation before any action has been performed is shown by *s0*. The new situation that results from the performing action *a* in situation *s* is denoted by *do(a,s)*. For instance, in the example of a robot world, if the robot's first action is *move(right)*, the resulting situation would be *do(move(right), s0)*. If the next action were *open(Door)*, the resulting situation would then be *do(open(Door), do(move(right), s0))*.
- *Fluents* are predicates and functions whose values may vary from situation to situation. Fluents can be seen as properties of the world. If we use the same robot example above, we define a fluent *isOpen(o,s)* which indicates that the robot has opened an object *o* in a particular situation. If the robot has opened the door after one step from the initial situation, we can observe that *isOpen(Door,do(open(Door),s0))* is true.

GOLOG [23, 35] is an example logic programming system that has been developed based on the situation calculus. A disadvantage of the situation calculus for robotic planning is that the knowledge of the current state is represented indirectly via initial conditions and actions which the agent has performed up to now. As a consequence, each time a condition is evaluated in an agent program, the entire history of actions is involved in the computation.

2.1.2 Fluent Calculus

The fluent calculus is a variant of the situation calculus. It solves the frame problem [6], which exists in the situation calculus. The main difference between the situation calculus and the fluent calculus is that in the fluent calculus, situations are considered to be representations of states while in the situation calculus, situations are histories of actions instead of states. A symbol o is used to concatenate terms that represent facts which are in a situation. Situations which are changed after the execution of an action are removed and the rest stays the same. We now give an example on fluent calculus consisting of putting a box from the table to the shell. This example can be formalized as

$$State(Do(put(box,table,shell),s)) \circ on(box,table) = State(s) \circ on(box,shell) \ .$$

This formula states that after the action *put*, *on(box,shell)* term is added and *on(box,table)* term is removed.

Flux is an example fluent calculus implementation, yielding a programming method for the design of agents that reason logically about their actions and sensor information in the presence of incomplete knowledge [40, 39].

2.1.3 Partial Order Planning

Plans can be represented as *partial orders*, in which some steps are ordered with respect to others, but other steps are left unordered. The planner starts with an initial plan representing the *start* and *end* steps. Afterwards, in each iteration, the planner adds one more step. If one branch of the search space leads to an inconsistent plan, the planner backtracks and tries another branch.

2.1.4 Total Order Planning

An alternative to *partial order planning* is *total order planning*. In this approach, plans consist of a simple list of steps. In total order planning, every step is

ordered with respect to every other step. As a conclusion, all possible steps are specified at the beginning of the plan. Since there are no unordered steps, neither non-determinism nor backtracking exists in a plan.

2.2 Logic

In mathematics, we use logic in a theoretical manner. However, in computer science, we are interested in using logic in practice. That is why there are many kinds of logic in philosophy and computer science, while in mathematics, classical first-order logic is usually sufficient to formalize correct principles of mathematical reasoning. Another important difference between traditional mathematics and computer science is that; "truth" exists abstractly in mathematics, independently of anyone knowing the truth or the falsehood of a proposition, while in the computer science, proofs show how to construct objects. For example, $\exists.A(x)$ is true if we can construct an object t such that $A(t)$ is true. Also $A \supset B$ is true if we can construct a proof of B from a proof of A.

Since we consider intuitionistic logic (also called constructive logic) in this thesis, we need to mention some differences between classical logic and intuitionistic logic. In classical logic, the principle of excluded middle ($A \vee \neg A$) is considered to be true, while in intuitionistic logic, it is rejected. Also, intuitionistic logic rejects proof by contradiction because we need to verify the proposition itself rather than just falsifying its negation. Similar to this, the double negation of a proposition is not equal to itself ($\neg\neg A \neq A$) in intuitionistic logic.

2.2.1 Propositional Logic

In propositional logic,we study the behavior of propositional connectives, such as 'and' and 'or' [10, 20]. It is assumed that there is a family of sentences that can be thought of as expressing primitive propositions. For instance, the English sentence "Milk is white" expresses the proposition that milk is white. We represent

each such primitive proposition by a single letter like P,Q, ... One of the most important things is that each proposition is either true or false, but can not be both.

A formula in propositional logic is a sentence built up from propositional letters and propositional connectives. Propositional connectives can be constants (0-place) like *'T'* and *'⊥'*, for true and false, unary like *'¬'* for negation, binary (2-place), ternary (3-place) like "if-then-else", and so on. The most commonly used connectives are ∨ for or, ∧ for and, and ⊃ for implication. If we use these connectives, we can form an example proposition like

$$p \wedge q \supset \neg r \vee q \ ,$$

which means that "if p and q then not r or q". For the sake of clarity, it is more appropriate to write the above formula like

$$(p \wedge q) \supset ((\neg r) \vee q) \ .$$

2.2.2 Natural Deduction

There are some theorem proving mechanisms such as tableaux, resolution, Hilbert system [14], sequent calculus, natural deduction, etc. In this section, we will briefly describe Natural Deduction. Natural deduction attempts to provide a deductive system which is a formal model of logical reasoning as it naturally occurs. Natural deduction's modern form was independently proposed by the German mathematician Gentzen in 1935 [11].

In natural deduction, we have a collection of proof rules (inference rules), which allow us to reach a conclusion given a certain collection of premises. Each logical connective and quantifier is defined by one or more introduction rules, specifying how to infer a connective. There are also elimination rules which tell what information can be deduced from the presence of a compound proposition.

For example, the introduction rule for the conjunction is

$$\frac{A\ true \qquad B\ true}{A \wedge B\ true}\ \wedge\mathtt{I}\ ,$$

asserting that if we have two premises 'A *true*' and 'B *true*', then we can derive the judgment '$A \wedge B$ *true*' in the conclusion.

The elimination rules for conjunction are

$$\frac{A \wedge B\ true}{A\ true}\ \wedge\mathtt{E}_L \qquad\qquad \frac{A \wedge B\ true}{B\ true}\ \wedge\mathtt{E}_R\ .$$

From conjunction elimination rules, we can derive the judgments 'A *true*' and 'B *true*' separately with the elimination of the judgment '$A \wedge B$ *true*'.

We continue with the introduction rules of the disjunction as

$$\frac{A\ true}{A \vee B\ true}\ \vee\mathtt{I}_L \qquad\qquad \frac{B\ true}{A \vee B\ true}\ \vee\mathtt{I}_R\ ,$$

meaning that if we know 'A true', we can conclude 'A $\vee$ B true' and also if we know 'B true', we can conclude 'A $\vee$ B true'.

The elimination rule for the disjunction is

$$\frac{A \vee B\ true \quad \begin{array}{c}[A] \\ \vdots \\ C\end{array} \quad \begin{array}{c}[B] \\ \vdots \\ C\end{array}}{C\ true}\ \vee\mathtt{E}\ ,$$

which means that if we have $A \vee B$, and we can prove C from both A or B when assumed alone, we can conclude that C is true.

The introduction rule for the implication is

$$\frac{\begin{array}{c} A \\ \vdots \\ B \end{array}}{A \supset B \; true} \supset \mathtt{I} \; .$$

The introduction rule for implication means that, if we can prove B from the knowledge of A, then we can say that $A \supset B$ is true.

The elimination rule for the implication is

$$\frac{A \; true \quad A \supset B \; true}{B \; true} \supset \mathtt{E} \; ,$$

which means that if we know A is true and also that $A \supset B$ is true, we can conclude that B is also true.

2.2.3 First-Order Logic

Propositional logic deals with the sentence components like *not*, *and*, *or* and *if ... then*, but the logical aspects of the natural and the artificial languages are much richer than that [37, 38, 20]. What can we do if we want to use sentence components like *there exists*, *all*, *among* and *only* ?

First-order logic (or predicate logic) is an extension of propositional logic in which formulas may contain variables that can be quantified. Variables are written with lovercase letters *u,v,w,x,y,z....* Two common quantifiers are the existential $\exists$ and the universal $\forall$ quantifiers. For example, if we wanted to express the statement

"For every x, if x is a student, then there is some y which is an instructor such that x is younger than y"

in first-order logic, we can formalize the sentence as

$$\forall \mathrm{x}(\mathrm{S}(\mathrm{x}) \supset (\exists \mathrm{y}(\mathrm{I}(\mathrm{y}) \wedge \mathrm{Y}(\mathrm{x},\mathrm{y})))) \; ,$$

where

S(x): x is a student

I(x): x is an instructor

Y(x,y): x is younger than y.

2.2.4 Intuitionistic First-Order Linear Logic

Linear logic was first introduced in [13] by Girard and can be interpreted in the scope of classical or intuitionistic logic. Our approach to linear logic is in an intuitionistic way. Intuitionistic Linear Logic (*ILL*) is a refinement of intuitionistic logic where formulae must be used exactly once (weakening and contraction rules [13] are removed). *ILL* is a resource sensitive logic because assumptions can be consumed during inference. *ILL* can be used to formalize planning problems in a way that elegantly solves the frame problem [6]. The resource consumption property in *ILL* also provides a more expressive language for planning. For example, new state elements can be created or deleted and also non-deterministic and sensing actions can be expressed in *ILL*. Each proof in intuitionistic logic directly corresponds to a program, so it is possible to transform each proof into an executable plan.

For the motivation of linear logic, the Blocks World [16] can be given as an example which is often used in artificial intelligence and planning problems. In this domain, there is a set of blocks on the table and a robot arm can pick up blocks and place them either on the table or on the another block. The goal is to have vertically ordered blocks whose order is initially given.

First, we choose the collection of propositions shown in Table 2.1 to encode the blocks world example.

We can then define an initial state which, for example, specifies that block c is on block b, block b and block a are on the table and the robot arm is empty.

empty	robot arm is empty
on(x, y)	block x is on block y
tb(x)	block x is on the table
clear(x)	the top of block x is clear
holds(x)	robot arm holds block x

Table 2.1: Collection of propositions for the Blocks World

This state can be described as

$$\Delta_0 = (\mathit{empty},\ \mathit{tb}(a),\ \mathit{tb}(b),\ \mathit{clear}(a),\ \mathit{clear}(c)\ ,\ \mathit{on}(c, b)).$$

We can also describe the goal state to be achieved as a logical proposition using the same set of propositions such as on(b,c). But how we can describe the legal moves to achieve the goal? With an example, we can show some approaches.

Consider this example:

If the robot hand is empty, a block x is clear, and x is on y, then we can pick up the block, that is, achieve a state where the robot hand holds x and y is clear.

If one tries to use a logical implication as an action to formulate this example, the formulation would be

$\forall$x.$\forall$y.(empty $\wedge$ clear(x) $\wedge$ on(x,y)) $\supset$ (holds(x) $\wedge$ clear(y)) .

However, using this sentence as an hypothesis and putting c for x and b for y, we can derive contradictory propositions like empty $\wedge$ holds(c). So, using just logical implication for this formulation is incorrect.

If we try to solve this problem in temporal logic [25, 12] using a notion of time 'O', where OA means truth of A at the next time, then we can write

$\forall$x.$\forall$y.(empty $\wedge$ clear(x) $\wedge$ on(x,y)) $\supset$ O(holds(x) $\wedge$ clear(y)) .

The contradiction problem can be then solved. However, we need to also express that everything else stays the same when we pick up a block. This

suggests us that we do not need a logic of time but a logic of state. Fortunately linear logic gives a good solution to these kind of problems. We implement blocks world example in 5.4.2. The general form of a linear hypothetical judgment is

$$A_1\ true, ..., A_n\ true \Vdash C\ true\ ,$$

which means that using every assumption exactly once, we can prove C from assumptions $A_1, ..., A_n$. In this judgment, only linear hypotheses can be used. We can also introduce a new judgment in order to accommodate ordinary intuitionistic or classical reasoning. It is called validity, encoding unrestricted hypotheses in addition to linear resources. Linear assumptions can be seen as consumable resources and unrestricted assumptions can be seen as non-consumable resources or rules of the theorem that we can use unlimited times.

Together with unrestricted resources and linear resources, we can write a new judgment as

$$(v_1 : B_1\ valid, ..., v_m : B_m\ valid); (u_1 : A_1\ true, ..., u_n : A_n true) \vdash C\ true\ .$$

We separate the two different types of assumptions by a semicolon ";". Furthermore, we abbreviate unrestricted assumptions by Γ and linear assumptions by Δ. We show the judgment in a short form as

$$\Gamma; \Delta \Vdash C\ true.$$

2.2.4.1 Connectives in Linear Logic

In this section, we will briefly introduce important connectives of linear logic.

- Simultaneous Conjunction ($\otimes$):

  $A \otimes B$ is pronounced as "A and B" or "A tensor B". If both A and B are true in the same state, then we can write $A \otimes B$. For an example, assume

that we have 2 Euros and we want to buy a coffee (1 Euro) and a chocolate (1 Euro). Since we have enough money for both coffee and chocolate, we can buy both them at the same time. After buying, we can conclude that $coffee \otimes chocolate$ is true and there is no money left.

- Alternative Conjunction (&):

  $A\&B$ is pronounced as "A with B" or sometimes called *internal choice*. If we can conclude A using some resources Δ and also we can conclude B with the same resources Δ, we can write $A\&B$. For instance, we have 1 Euro and the price of a coffee and a tea is 1 Euro. So we can say that $coffee \ \& \ tea$ which means that using 1 Euro, we can buy a coffee or a tea, not both at the same time.

- Linear Implication ($\multimap$):

  $A \multimap B$ is pronounced as "A linearly implies B" or "A lolli B". If we can achieve B from A, then we can write $A \multimap B$. We must note that A must be used exactly once to properly implement linearity.

- Disjunction ($\oplus$):

  $A \oplus B$ is also called *external choice*. Our resources can make either A or B true, then we can write $A \oplus B$.

- Unit (1):

  The goal 1 can always be produced without using any resources. This is the identity for simultaneous conjunction with $1 \otimes A = A$.

- Top (T):

  The goal T can always be achieved regardless of any available resources. It always consumes all of the resources. T is the identity for alternative conjunction, with $T\&A = A$.

- Impossibility (0):

  If have some resources and they conclude to 0, then we can conclude anything from that resources and with other resources. However, we have $0 \oplus A = A$.

- "Of Course" Modality (!):

  The unary operator ! connects unrestricted hypotheses and linear hypotheses. For example, if we have unlimited resource of coffee, we can represent this expression as $!coffee$.

The usage of these connectives can be shown with a mobile robot example.

Robot is *At(0)* point and it's initial power is full	(P(100) $\otimes$ At(0)) $\multimap$
Moves 100mt. forward	At(100)
Moves 100mt. backward	& At(-100)
Moves forward 50mt. and *Launches Missile*	& (At(50) $\otimes$ LM)
With additional 50 Power *Launches Missile* or the robot *Charges* itself with the solar energy if the weather is sunny	& ((P(50) $\multimap$ LM) $\oplus$ (S $\multimap$ C))

Table 2.2: A Mobile Robot and Linear Logic Encoding

The example on Table 2.2 tells us that with the full power, what the mobile robot can do. In this example, the robot can choose only one of the actions to do. We also observe that, with additional 50 power, the robot can launch missile. Or, the robot can charge itself if the weather is sunny.

2.2.4.2 Sequent Calculus for Linear Logic

Natural deduction is not well-suited for proof search, because it involves mixing forward and backward reasoning even if we restrict ourselves to searching for normal deductions. Flow of information in the elimination rules is downwards and flow of information in the introduction rules is upwards. We need a deterministic mechanism to find derivations for a given proposition. In this section, we describe a sequent calculus as a calculus of proof search for normal natural deductions [31].

Sequent calculus flips elimination rules in natural deduction to be used in an upside-down manner. With this modification, proof search proceeds only bottom-up [32]. This modification turns the introduction rules into *right rules* and the elimination rules into *left rules*. In sequent calculus, we denote a judgment by

$$A_1, ..., A_n \Longrightarrow A,$$

where propositions A_1 to A_n, at the left side of the arrow, are assumptions and A, at the right side of the arrow, is called the goal. In Appendix A, we give sequent rules for intuitionistic first-order linear logic on figures A.1, A.2, A.3, A.4 and A.5.

2.3 Constraint Logic Programming

Constraint logic programming was first introduced in 1987 [22]. Constraint logic programming is an extension of constraint programming, in which logic programming is extended to include concepts from constraint satisfaction methods [27, 21]. A constraint logic program may contain constraints in the body of clauses. The following is an example of a constraint inside a clause.

$$A(X, Y) \text{ :- } X + Y > 0,\ B(X),\ C(Y)\ ,$$

where $X+Y > 0$ is a constraint, A(X,Y), B(X), and C(Y) are literals as in regular logic programming. This clause tells us that A(X,Y) holds if X+Y is greater than zero and both B(X) and C(Y) are true.

A proof for a goal is composed of clauses and literals. Clause bodies are formed with satisfiable constraints and literals can in turn be proved using other clauses. Execution starts from the goal and recursively scans clauses, trying to prove the goal. Constraints encountered during this scan are placed in a set called *constraint store*. If this set is found to be unsatisfiable, the interpreter backtracks and tries to use other clauses for proving the goal.

Semantics of constraint logic programs can be defined as a pair $< G, S >$ for an interpreter. The first element of the pair, G, is the current goal, and the second element, S, is the constraint store. The current goal contains literals that the interpreter is trying to prove and may contain some constraints to satisfy that it is trying. The constraint store contains all constraints that the interpreter thinks are satisfiable.

At first, G contains the current goal and S is empty. The interpreter removes the first element from the current goal and begins to analyze it. The result of this analysis is either a failure or a successful termination. During the analysis, some new literals may be added to the current goal or some constraints may be added to the constraint store. The addition of the constraints to the constraint store may cause constraints in the constraint store to become unsatisfiable. If there is a condition such as the unsatisfiability of constraints, the interpreter backtracks to a position where the constraints can be satisfied. The main goal is achieved if the current goal is empty and all the constraints in the constraint store are satisfiable.

2.4 Proof Search In Linear Logic

Proof search can be simply described as finding a proof for a given theorem. Finding proofs is more challenging than merely proving theorems, since proofs contain more information than the theorems they prove. Proof search in linear logic can have a variety of applications depending on the problem. Thus, searching proofs in such an expressive logic is difficult. For instance, if we search for a proof in the domain of planning problems, that means we search for a plan. Or, if we search for a proof in the domain of functional programming and type theory, that means we search for a program satisfying a given specification.

In proof search, we may need different requirements for each application. However there are some basic techniques that are applicable to almost all applications. We point out some of such basic techniques in this section.

2.4.1 Bottom-up Proof Search

As in [31, 4], we define buttom-up proof search as starting with a given goal sequent and using inference rules of the logical system in the backward direction in order to refine goals until we are left with axiomatic or initial sequents. At any time in bottom-up search, after applying some inference rules, we have a partial derivation with undecided judgments. The goal is to derive all remaining judgments to complete a proof. We proceed by selecting a judgment which remains to be derived and an inference rule with which it might be inferred from. We may also need to determine exactly how the conclusion of the rule matches the judgment.

2.4.2 Unification

Unification in logic programming is the problem of binding the contents of variables, atoms or terms. Herbrand first introduced unification [17]. Afterwards, in [36], Alan Robinson introduced a more detailed formulation of unification for automated deduction.

When proving a proposition of the form $\exists x.A$ by its right rule in sequent calculus, we must supply a term t and then prove $[t/x]A$. When the domain includes infinitely many terms, we can not try all possible terms. However, we can postpone the choice of t and substitute a new variable X, an *existential variable*, for x in A. Finding an instantiation for existential variables under which two propositions or terms match is called unification. It's purpose is to eliminate existential non-determinism.

Unfortunately, unification with parameters is not so easy to handle. For instance, $\forall x.\exists y.y = x$ is valid, while $\exists y.\forall x.y = x$ is not [32]. We show each steps

of the latter:

$$\begin{aligned} \exists y.\forall x.y = x & \\ \forall x.Y = x & \quad (\exists I) \\ Y = a & \quad (\forall I) \\ \# & \end{aligned}$$

In this derivation, we postpone choosing the instantiation for y by supplying an existential variable in the rule $\exists I$. Afterwards, we put a parameter a for x. The parameter a is a fresh variable and can not exist before. At the last step, we check if Y may or may not be instantiated with a parameter a. For this control, we use *Skolemization* which is described in 5.2. In this example, we can say that the existential variable Y is created before the parameter a. Therefore, Y can not be instantiated with the parameter a.

2.5 Focused Intuitionistic First-Order Linear Logic (FocLL)

Existing efficient implementations of linear logic, such as the Lolli language [18], often restrict their language to *Linear Hereditary Harrop Formulas* (LHHF) to simplify proof search and ensure determinism in the proof search to support logic programming. These languages correspond to those that are freely generated by the grammar

$$A := p \mid A \multimap A \mid A \,\&\, A \mid T \mid A \supset A \mid \forall x.A,$$

as well as positive occurrences of other linear connectives [31]. Unfortunately, unsupported negative occurrences of missing connectives may potentially be useful to capture nondeterministic state components the for robotic planning problems and we would like to have a logical system that allows us to experiment with the full language. We would like to use a grammar that does not impose such a restriction and incorporates all linear connectives. We will address the question of whether such an expressive grammar is needed for realistic problems later in this thesis, where we describe specific planning problems. In the meantime, the

language we consider in this section and subsequently for our proof system is given by

$$A := P \mid A \multimap A \mid A \,\&\, A \mid T \mid A \supset A \mid A \otimes A \mid 1 \mid A \oplus A \mid 0 \mid !A \mid \forall x.A \mid \exists x.A \,,$$

where P ranges over atomic formulas having the form $p(t_1, ..., t_n)$, defined according to the specifics of a particular domain.

Efficient methods for theorem proving rely on a classification of connectives based on the invertibility of associated left and right sequent calculus rules [31]. We adopt a similar classification for our language and the associated proof theory in order to guide proof search through proper focusing choices:

- Atomic : P
- Right Asynchronous : A1 $\multimap$ A2, A1 & A2, T, A1 $\supset$ A2, $\forall$ x.A
- Left Asynchronous : A1 $\otimes$ A2, 1, A1 $\oplus$ A2, 0, !A, $\exists$ x.A
- Right Synchronous : A1 $\otimes$ A2, 1, A1 $\oplus$ A2, 0, !A, $\exists$ x.A
- Left Synchronous : A1 $\multimap$ A2, A1 & A2, T, A1 $\supset$ A2, $\forall$ x.A ,

where the terms *asynchronous* and *synchronous* denote whether associated rules are invertible or not, respectively.

In this context, the nondeterminism associated with synchronous occurrences of certain connectives presents serious problems for logic programming systems where operational semantics must be unambiguously defined. Even though this nondeterminism does not present a fundamental problem for our domain (where the presence of multiple different proofs for a single sequent simply corresponds to alternative solutions for a planning problem), it does impact the efficiency of the resulting system. Consequently, we also seek to eliminate as much nondeterminism as possible from the proof theory while preserving completeness with respect to the semantics of the original intuitionistic linear logic. Similar to the methods described in [31], we classify nondeterminism in five different categories:

- **Conjunctive choices**: The order in which subgoals of a rule are attempted is usually left unspecified by the proof theory. This is a form of *don't care nondeterminism*.

- **Disjunctive choices**: When multiple disjunctive alternatives are available, the order in which they are attempted is not important in the absence of side-effects. This is a form of *don't-know non-determinism* and necessitates back-tracking.

- **Resource choices**: For multiplicative connectives, different ways in which available resources can be divided among parallel goals is another significant source of don't know nondeterminism. Proper resource management and delaying of associated decisions can solve this kind of determinism.

- **Universal choices**: The choice of fresh parameters within ∀R and ∃L rules leads to another form of don't-care non-determinism.

- **Existential choices**: The need for choosing specific terms to replace the quantified variable within the ∃R and ∀L rules leads to another source of don't-know non-determinism. This is usually addressed by unification and its variants that delay such decisions until sufficient information is available.

2.5.1 Focusing

Focusing in linear logic was first introduced by Andreoli [24]. In this section, we describe the intuitionistic formulation presented in [31], which we call FocLL[1], on which our proof theory will be based. Focusing is used to eliminate non-determinism which occurs as a result of disjunctive choices in proof search while maintaining the soundness and completeness of the proof theory. Focusing has two main phases, inversion and focusing, alternating through decision rules.

[1]FocLL stands for Focused Intuitionistic First-Order Linear Logic.

2.5.1.1 The Inversion Phase

In focusing systems, right and left invertible connectives are eagerly decomposed during the inversion phase of the proof search. Normally, the order in which invertible rules is applied does not effect soundness or completeness of the proof system. However, an ordered context Ω is used to eliminate the associated don't-care nondeterminism. Judgments for right invertible and left invertible rules are defined as

$$\Gamma; \Delta; \Omega \Longrightarrow A \Uparrow ,$$

$$\Gamma; \Delta; \Omega \Uparrow \Longrightarrow C ,$$

where we have,

Γ : Unrestricted hypotheses (which may be arbitrary),
Δ : Linear hypotheses (not left asynchronous),
Ω : Ordered hypotheses (which may be arbitrary),
A : The goal (which may be arbitrary),
C : The goal (not right asynchronous).

- **Right Inversion Phase**

This phase is the entry point of the proof search, and is defined by the inference rules listed in Appendix B.1.

Note that the inversion phase proceeds until there are no right invertible rules left, foc-$\Uparrow$R rule is used to proceed with left invertible connectives.

- **Left Inversion Phase**

Once all right asynchronous connectives are eliminated, we proceed with the elimination of left asynchronous connectives in Ω. Recall that by construction, Δ

is not permitted to contain any left asynchronous connectives, so the consideration of Ω is sufficient for this phase. All left-invertible rules are listed in Appendix B.2.

When we encounter a proposition in Ω which is not left asynchronous, we use the $\Uparrow L$ rule and move it into Δ for later consideration during focusing. The inversion phase is concluded when no propositions are left in Ω. We then proceed with the decision phase.

2.5.1.2 Decision

When the decomposition of all left and right asynchronous connectives is completed, the goal is no longer asynchronous and the input context contains no left asynchronous propositions. At this point, we need to make a decision and choose a proposition to focus on. The rules associated with this phase are given in Appendix B.3. Two judgments for focusing on right and left are defined as

$$\Gamma;\, \Delta;\, \cdot \Longrightarrow C \Downarrow ,$$

$$\Gamma;\, \Delta;\, A \Downarrow \Longrightarrow C ,$$

where we have,

Γ : Unrestricted hypotheses (which may be arbitrary),
Δ : Linear hypotheses (not left asynchronous),
A : The focus proposition (which may be arbitrary),
C : The succedent (not right asynchronous).

2.5.1.3 The Focusing Phase

Once a decision is made, proof search proceeds by focusing on a specific non-invertible proposition and decomposing it until either an invertible connective or

an atomic proposition is reached. In the former case, proof search goes back to the inversion case, whereas the latter case terminates with the application of the init rule.

- **Right Focusing**

Appendix B.4 details inference rules related to right synchronous connectives. The right focusing terminates when we encounter a right asynchronous connective. In that case, proof search shifts back to the inversion phase and continues to decompose the right side of the sequent. Note that the R rule is applied when A is atomic as well, going directly to another decision phase to proceed with left focusing or the decomposition of an unrestricted resource.

- **Left Focusing**

Left focusing rules are listed in Appendix B.5. The **init** rule, as usual, is where unification will be done and resources that are left unused are shifted to the output context. Note also that if the focused proposition becomes asynchronous, we immediately switch back to the inversion phase and decompose the same proposition further.

In this chapter, we first gave some background for planning in robotics. We categorized planning as being done either in the space of situations or in the space of plans. Situation space planners are situation calculus and fluent calculus, plan space planners are partial order planning and total order planning. Afterwards, we gave background on types of logic such as propositional logic, first-order logic and intuitionistic first-order linear logic. We continued with describing constraint logic programming and proof search in linear logic. At the last section, we described Focused Intuitionistic First-Order Linear Logic (FocLL) presented in [31]. In subsequent chapters, we will use elements from this background to describe our contributions to build a theorem prover for robotic planning problems.

Chapter 3

Resource Management System For FOCLL (FRM)

First of all, we must note that our contributions begin with this chapter. In proof systems for linear logic, the need for resource management arises from non-deterministic decisions necessary to split resources in the backward application of the rules **foc-**$\otimes$R and **foc-** $\multimap$L. Since no further information is available in this formulation, all possible alternatives (2^n in the worst case) must be exhaustively searched. Fortunately, this problem can be solved using the *IO model* introduced in [19], where subgoal judgments only partially consume resources in their input Δ_I, returning unused resources as their output Δ_O.

We can eliminate a significant amount of non-determinism in splitting resources among parallel goals using the method in [19]. However, an additional problem remains in that, the presence of the logical constant T on the right hand side of a sequent allows the consumption of an arbitrary number of input resources. This problem can be solved by introducing an additional flag into the sequent, recording the presence of such a flexible resource sink to be considered by later stages of the search. This idea was introduced and developed in the context of linear logic programming in [3].

We introduce in this chapter, a new proof system for the full linear logic with

resource management and focusing, which we call *Full Resource Management System* (*FRM*). Our system is different from the *IO model* introduced in [19] since they restrict their language to *Linear Hereditary Harrop Formulas* (*LHHF*) to simplify proof search but our grammar incorporates all linear connectives.

3.1 Focusing

We use the same categorization of rules for the *FRM* system as we did for the FocLL system in Section 2.5.1. All rules are categorized according whether they are invertible or non-invertible. Sequent calculus rules for the *FRM* system are given in Figures 3.1, 3.2, 3.3, 3.4 and 3.5, corresponding to the right inversion, left inversion, decision, right focusing and left focusing phases of the proof search, respectively.

3.1.1 The Inversion Phase

We define judgments for right and left invertible rules as

$$\Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \;\; A \Uparrow ,$$

$$\Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Uparrow \Longrightarrow_v \;\; C ,$$

where

Γ	:	Unrestricted hypotheses (which may be arbitrary),
Δ_I	:	Input resources that may be consumed (not left asynchronous),
Δ_O	:	Output resources that are not consumed (arbitrary),
Ω	:	Ordered hypotheses (arbitrary),
A	:	The goal (arbitrary),
C	:	The goal (not right asynchronous),
v	:	Flag to encode freedom in resource consumption (0 or 1).

If any resources are in Ω, we have to use that resource to achieve the goal. In the $\multimap R_v$ rule, A must be used to achieve B since A is put into the ordered hypotheses context. This property is presented in the subcontext property which we will give details in the later section.

- **Right Inversion Phase**

Rules associated with this phase are given in Figure 3.1. In the *FRM* system, four variants are introduced for the $\&R$ rule to handle possible combinations of resource flags. Lack of flexibility in the resource consumption for at least one of the subgoals (v = 0) requires the exact presence of the associated output in the conclusion sequent as well. Only when both subgoals are flexible in the resource consumption can the output freely discard mismatches in the outputs.

- **Left Inversion Phase**

Inference rules for the left invertible rules are listed in Figure 3.2. The $\oplus L$ rules have four variants much like the $\&R$ rules. It is important to note that, in the left transition rule $\Uparrow L$, if A is not used and it is transmitted into the output context, the T-flag is set to 1 (v = 1). We claim that all resources in Ω are going to be used. However, if the T-flag is set to 1, that means there is a possibility of transmission A into the output contex. Thus, we need the rule $\Uparrow L_{1A}$. In contrast, if the T-flag is set to 0, we have to use all resources in Ω.

3.1.2 Decision

Decision rules in *FRM* are listed in Figure 3.3. Propositions in Γ are not guaranteed to be left synchronous in the rule *decide*L!. Therefore, it would not be possible to directly start focusing on them. To alleviate this problem, we first invoke the inversion stage on the selected proposition, the result of which will be presented to yet another invocation of the decision phase.

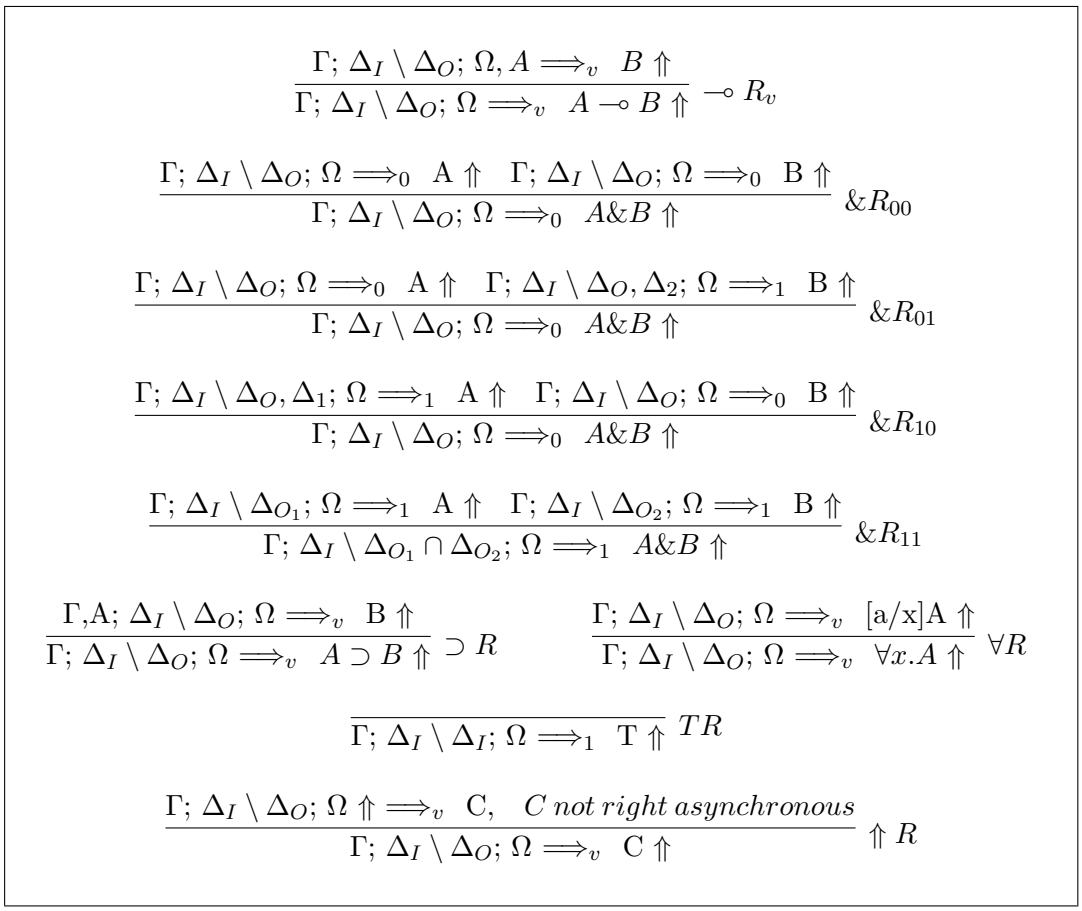

Figure 3.1: *FRM*, Right invertible rules

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A, B \Uparrow \Longrightarrow_v\ C}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \otimes B \Uparrow \Longrightarrow_v\ C}\ \otimes L$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ C \quad \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega, B \Uparrow \Longrightarrow_1\ C}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ C}\ \oplus L_{01}$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O, \Delta_1;\ \Omega, A \Uparrow \Longrightarrow_1\ C \quad \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_0\ C}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ C}\ \oplus L_{10}$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ \mathrm{C} \quad \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_0\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ \mathrm{C}}\ \oplus L_{00}$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_{O_1};\ \Omega, A \Uparrow \Longrightarrow_1\ \mathrm{C} \quad \Gamma;\ \Delta_I \setminus \Delta_{O_2};\ \Omega, B \Uparrow \Longrightarrow_1\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega, A \oplus B \Uparrow \Longrightarrow_1\ \mathrm{C}}\ \oplus L_{11}$$

$$\frac{\Gamma,\mathrm{A};\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ C}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, !A \Uparrow \Longrightarrow_v\ C}\ !L$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, [a/x]A \Uparrow \Longrightarrow_v\ C}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, \exists x.A \Uparrow \Longrightarrow_v\ C}\ \exists L$$

$$\frac{}{\Gamma;\ \Delta_I \setminus \cdot;\ \Omega,0 \Uparrow \Longrightarrow_0\ \mathrm{C}}\ 0L \qquad \frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega,1 \Uparrow \Longrightarrow_v\ \mathrm{C}}\ 1L$$

$$\frac{\Gamma;\ \Delta_I, A \setminus \Delta_O, A;\ \Omega \Uparrow \Longrightarrow_1\ C, \quad A\ not\ left\ asynchronous}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_1\ C}\ \Uparrow L_{1A}$$

$$\frac{\Gamma;\ \Delta_I, A \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ C, \quad A\ not\ left\ asynchronous}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_v\ C}\ \Uparrow L_v$$

Figure 3.2: *FRM*, Left invertible rules

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ C \Downarrow,\quad C \textit{ not atomic}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ C}\ \textit{decideR}$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Gamma;\ \Delta_I, A \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \mathrm{C}}\ \textit{decideL}$$

$$\frac{\Gamma, A;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Gamma, A;\ \Delta_I \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \mathrm{C}}\ \textit{decideL!}$$

Figure 3.3: *FRM*, Decision rules

3.1.3 The Focusing Phase

Note that the possibility of going back to the inversion phase distinguishes this system from the application of focusing and resource management to the *LHHF* language. For this limited language, focusing always either succeeds or fails, significantly reducing the impact of backtracking and increasing efficiency. Two judgments for focusing on right and left are defined as

$$\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ A \Downarrow\ ,$$

$$\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ C\ ,$$

where we have,

Γ	:	Unrestricted hypotheses (which may be arbitrary),
Δ_I	:	Input resources that may be consumed (not left asynchronous),
Δ_O	:	Output resources that are not consumed (arbitrary),
C	:	The goal (not right asynchronous),
A	:	The focus proposition (arbitrary),
v	:	Flag to encode freedom in resource consumption (0 or 1).

- **Right Focusing**

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_M;\ \cdot \Longrightarrow_v\ \mathrm{A} \Downarrow \quad \Gamma;\ \Delta_M \setminus \Delta_O;\ \cdot \Longrightarrow_w\ \mathrm{B} \Downarrow}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_{v \vee w}\ \mathrm{A} \otimes \mathrm{B} \Downarrow}\ \otimes R$$

$$\frac{\Gamma;\ \cdot \setminus \cdot;\ \cdot \Longrightarrow_v\ \mathrm{A} \Uparrow}{\Gamma;\ \Delta_I \setminus \Delta_I;\ \cdot \Longrightarrow_0\ !\mathrm{A} \Downarrow}\ !R \qquad \frac{}{\Gamma;\ \Delta_I \setminus \Delta_I;\ \cdot \Longrightarrow_0\ 1 \Downarrow}\ 1R$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \Downarrow}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \oplus \mathrm{B} \Downarrow}\ \oplus R_1 \qquad \frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{B} \Downarrow}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \oplus \mathrm{B} \Downarrow}\ \oplus R_2$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ [\mathrm{t/x}]\mathrm{A} \Downarrow}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \exists \mathrm{x.A} \Downarrow}\ \exists R$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \Uparrow \textit{ not right synchronous}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \Downarrow}\ \Downarrow R$$

Figure 3.4: *FRM*, Right focusing rules

Inference rules for the focusing rules are listed in Figure 3.4. The $\otimes R$ rule is one of the rules where resource management is implemented. Resources left unused by the first subgoal are shifted to the second subgoal. The resource flexibility flags resulting from both subgoals are combined with a logical or, since simultaneous conjunction can push unused resources to either one of the subgoals if they happen to be flexible in their resource consumption.

An important observation is that Δ_M, leftover resources from the first subgoal will never contain any left asynchronous formulas. This is because Ω starts out empty for the right focusing phase, making it impossible to shift any such connectives to the output. This guarantees that the input resources to the second subgoal are all left synchronous as well.

- **Left Focusing**

Left focusing rules are presented in Figure 3.5. Similar to the $\otimes R$ rule, the $\multimap L_1$ rule uses resource management to increase efficiency. Once again, we can

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_M;\ \cdot \Longrightarrow_v\ A \Downarrow \quad \Gamma;\ \Delta_M \setminus \Delta_O;\ B \Downarrow \Longrightarrow_w\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ A \multimap B \Downarrow \Longrightarrow_{v \vee w}\ \mathrm{C}}\ \multimap L$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ B \Downarrow \Longrightarrow_v\ \mathrm{C} \quad \Gamma;\ \cdot \setminus \cdot;\ \cdot \Longrightarrow_w\ A \Uparrow}{\Gamma;\ \Delta_I \setminus \Delta_O;\ A \supset B \Downarrow \Longrightarrow_v\ \mathrm{C}}\ \supset L$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ A \& B \Downarrow \Longrightarrow_v\ \mathrm{C}}\ \&L_1 \qquad \frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ B \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ A \& B \Downarrow \Longrightarrow_v\ \mathrm{C}}\ \&L_2$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ [t/x]A \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \forall x.A \Downarrow \Longrightarrow_v\ \mathrm{C}}\ \forall L$$

$$\frac{}{\Gamma;\ \Delta_I \setminus \Delta_I;\ \mathrm{P} \Downarrow \Longrightarrow_0\ \mathrm{P}}\ init$$

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \mathrm{A} \Uparrow \Longrightarrow_v\ \mathrm{C} \quad A\ not\ atomic\ and\ not\ left\ synchronous}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \mathrm{A} \Downarrow \Longrightarrow_v\ \mathrm{C}}\ \Downarrow L$$

Figure 3.5: *FRM*, Left focusing rules

guarantee that neither Δ_O, nor Δ_I will have any asynchronous connectives since the first subgoal has an empty Ω.

Key properties can be found in Appendix C.1, soundness can be found in Appendix C.2 and completeness of the *FRM* system can be found in Appendix C.3.

In this chapter, we introduced a new proof system for the full linear logic with resource management and focusing, which we call *Full Resource Management* (*FRM*). The *FRM* system solves the problem of consuming an arbitrary number of input resources in the case of the presence of T on the right hand side of a sequence, by introducing an additional flag into the sequent. At the end of the chapter, we proved the soundness and completeness of *FRM*.

Chapter 4

Adding Constraints

4.1 Adding Constraints Into Intuitionistic Linear Logic

Robot behaviors generally include nontrivial goals. Almost all robotic planning problems include dynamical continuous constraints. Modeling of only the discrete aspects of a problem may not be sufficient to achieve nontrivial goals. We also have to consider continuous aspects for these kind of goals. Since we would like to represent and reason about such problems in our logical system, the current nature of *Intuitionistic First-Order Linear Logic* is not sufficient to achieve our goals. To this end, we need to combine dynamical constraints with our *FRM* system.

In [41], *Constraint Intuitionistic Linear Logic* (*CILL*) is presented, merging continuous constraint solvers with linear logic. As a result, hybrid properties of robotic behaviors can be expressed and reasoned with. By using constraint solvers for particular domains, we can reduce the complexity associated with the encoding.

In order to formulate *CILL* using sequent calculus, we need a new context to collect and solve constraints. It is called the *constraint context* and denoted

with the symbol "Ψ". The new judgment, incorporating the constraint context, unrestricted hypotheses and linear resources is defined as

$$\Psi \mid \Gamma; \Delta \Longrightarrow A \ .$$

The meaning of this judgment is that, if constraints in Ψ are satisfiable, then using consumable resources Δ and unrestricted hypotheses Γ, we can achieve the goal A. In order to combine constraints into *Intuitionistic Linear Logic*, we need two new connectives, which we describe below.

Constraint Implication: The first connective is *constraint implication* which introduces a constraint precondition to a linear expression. The right and left rules for constraint implication are given as

$$\frac{(\Psi, D) \mid \Gamma; \Delta \Longrightarrow A}{\Psi \mid \Gamma; \Delta \Longrightarrow D \supset_c A} \supset_c R \qquad \frac{\Psi \models D \qquad \Psi \mid \Gamma; \Delta, A \Longrightarrow C}{\Psi \mid \Gamma; \Delta, D \supset_c A \Longrightarrow C} \supset_c L \ .$$

The right rule can be read as follows: If we want to achieve $D \supset_c A$, then we need to show that the goal A can be achieved under the constraint D and existing constraints. The left rule for this connective is very similar to typical left rules for implication, except that the constraint D has to be handled by another proof procedure specific to the constraint domain. $\Psi \models D$ means that the combination of all constraints in the constraint store must entail the constraint D.

Constraint Conjunction: The second connective for the *CILL* is *constraint conjunction*, which asserts the validity of a constraint in conjunction with a linear logic expression. The following rules are right and left rules for the constraint conjunction connective.

$$\frac{\Psi \models D \qquad \Psi \mid \Gamma; \Delta \Longrightarrow A}{\Psi \mid \Gamma; \Delta \Longrightarrow D \wedge_c A} \wedge_c R \qquad \frac{(\Psi, D) \mid \Gamma; (\Delta, A) \Longrightarrow C}{\Psi \mid \Gamma; \Delta, D \wedge_c A \Longrightarrow C} \wedge_c L \ .$$

Right rule can be interpreted as follows, if we want to show that $D \wedge_c A$ is achievable, then we need to show that both the goal A can be achieved with given resources and constraint D is valid under given constraints. The left rule asserts that the constraint D and consumable resource A are extracted from $\Psi \mid \Gamma; \Delta, D \wedge_c A \Longrightarrow C$ and are inserted into their own contexts.

Constraint Contradiction: The third connective is *constraint contradiction*. It is given as

$$\frac{\Psi \models \bot}{\Psi \mid \Gamma; \Delta \Longrightarrow C} \bot .$$

This rule asserts that if we have an inconsitent constraint domain, we can conclude that achieving any arbitrary goal.

Constraint Split: The last connective is *constraint splitting* and the associated rules are given as

$$\frac{\Psi \models \Psi_1 \vee \Psi_2 \quad \Psi_1 \mid \Gamma; \Delta \Longrightarrow C \quad \Psi_2 \mid \Gamma; \Delta \Longrightarrow C}{\Psi \mid \Gamma; \Delta \Longrightarrow C} \vee \mathit{split} ,$$

$$\frac{\Psi \models \exists x.\Psi_1(x) \quad \Psi_1(x) \mid \Gamma; \Delta \Longrightarrow C}{\Psi \mid \Gamma; \Delta \Longrightarrow C} \exists \mathit{split} .$$

These two rules are also needed to handle inconsistency of constraints. We give more information about the first rule in 4.2.1. The second rule handles possible existential nondeterminacy in the constraint domain.

4.1.1 An Example: The Balanced Blocks World

The Blocks World domain serves as a simple but rich testbed for planning algorithms and methods. However, its scope has been limited to discrete planning. In

order to illustrate the application of *CILL* to robotic planning problems, the Balanced Blocks World (BBW) is introduced in [41]. In the BBW, dynamic balance and physical alignment properties of planar blocks are also considered in conjunction with logical properties associated with different stackings. Top half of Table 4.1 defines some predicates which shows the current state of block placements. These predicates are used as linear resources. On the other hand, the bottom half of Table 4.1 gives invariant facts about the world such as the colorings and slot positions. These predicates are used as unrestricted hypotheses.

dynamic state of the system	
$\mathsf{tableempty}(i)$	There are no blocks on the table
$\mathsf{ontable}(b, i)$	Block b is directly on slot i of the table
$\mathsf{available}(b)$	Block b is available for placement
$\mathsf{on}(a, b, x)$	Block a is on top of Block b at an absolute position x
$\mathsf{clear}(b, x)$	Block b is at absolute position x and its top is clear
invariant facts about the world	
$\mathsf{tcol}(b, c)$	The top of block b has color c
$\mathsf{bcol}(b, c)$	The bottom of block b has color c
$\mathsf{slotcol}(i, c)$	Slot i on the table has color c
$\mathsf{slotisat}(i, x)$	Slot i is located at distance x from table origin

Table 4.1: Resource predicates for the Balanced Blocks World

After the definition of the BBW, we now give an example on the usage of *CILL* for planning in the BBW domain. First we give the starting state where there is a single empty slot on the table and also there are two available blocks a and b.

$$\Delta_0 = (\mathsf{tableempty}(1), \mathsf{available}(a), \mathsf{available}(b))$$

There are two kinds of unrestricted context. First one, Γ_f, includes logical formulae to capture invariant facts about the environment.

$$\Gamma_f = (\mathsf{tcol}(a, red), \mathsf{bcol}(a, blue), \mathsf{tcol}(b, blue), \mathsf{bcol}(a, grn), \mathsf{slotcol}(1, grn), \mathsf{slotisat}(1, 0))$$

The second unrestricted context includes models of actions that are available in the domain in the form of linear implications. In table 4.2, Γ_a is summarized.

putontable :	$\forall a.\forall i.\forall c.\forall x_i.available(a) \otimes tableempty(i) \otimes slotcol(i,c) \otimes bcol(a,c) \multimap ontable(a,i) \otimes clear(a)$
getofftable :	$\forall a.\forall i.ontable(a,i) \otimes clear(a) \multimap available(a) \otimes tableempty(i)$
putonblock :	$\forall a.\forall b.\forall c.\exists x_a.available(a) \otimes clear(b) \otimes bcol(a,c) \otimes tcol(b,c) \multimap on(a,b,x_a) \otimes testing(a) \otimes check(b, mass(a), xa)$
getoffblock :	$\forall a.\forall b.\exists x_a.on(a,b,x_a) \otimes clear(a) \multimap available(a) \otimes clear(b)$
checkiter :	$\forall a.\forall b.\forall m.\forall x_m.\forall x_a.check(a,m,x_m) \otimes on(a,b,x_a) \multimap isin(x_m - x_a, tleft(a), tright(a)) \supset_c \Big(check(b, m+mass(a), \frac{mx_m+mass(a)x_a}{m+mass(a)}) \otimes on(a,b,x_a)\Big)$
checkend :	$\forall a.\forall b.\forall m.\forall i.\forall x_a.\forall x_m.check(a,m,x_m) \otimes ontable(a,i) \otimes slotisat(i,x_a) \otimes testing(b) \multimap isin(x_m - x_a, tleft(a), tright(a)) \supset_c (ontable(a,i) \otimes clear(b))$

Table 4.2: CILL representations of BBW actions and supporting rules for checking balance of newly placed blocks

Now we can specify the goal as the final component for the planning problem. If our goal is to reach a state where block b is placed either on the table or on another block, we can express this goal as

$$G = (\exists i.ontable(b,i) \oplus \exists a.\exists x.on(b,a,x)) \otimes T.$$

We must point that T is used to specify incomplete goals since consumes all resources left unused by the rest of the proof. Disjunction connective is used to indicate that there are two alternative branches and proof construction has to pick which one of these will be satisfied. Now, we can express the planning problem as a sequent,

$$\Psi_c \mid (\Gamma_f, \Gamma_a); \Delta_0 \Longrightarrow G$$

where additional environmental constraints can be specified in Ψ_c.

4.2 Adding Constraints Into FRM (CFRM)

In this section, we will incorporate constraint rules of the *CILL* language into *FRM*. We call this new system *CFRM* with the associated judgments defined as

$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; C \Uparrow ,$$
$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Uparrow \Longrightarrow_v \; C ,$$
$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \cdot \Longrightarrow_v \; C \Downarrow ,$$
$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, A \Downarrow \Longrightarrow_v \; C ,$$

In these definitions, we have,

Ψ	: The constraint store,
Γ	: Unrestricted hypotheses ,
Δ_I	: Input resources that may be consumed ,
Δ_O	: Output resources that are not consumed ,
Ω	: Ordered hypotheses ,
C	: The goal ,
A	: The focused proposition ,
v	: Flag to encode freedom in resource consumption (0 or 1) .

In the previous chapter, we have proven that the *FRM* system is sound and complete. However, we have not yet proven that *CILL* system is complete and it is not trivial to prove.

Inference rules for the right invertible rules are listed in Figure 4.1, left invertible rules are listed in Figure 4.2, decision rules are listed in Figure 4.3, right focusing rules are listed in Figure 4.4 and left focusing rules are listed in Figure 4.5.

4.2.1 Restrictions with Constraints

Our CFRM rule set does not include two important aspects of constraint handling, case splitting and supporting for interpreted symbols during unification.

$$\frac{(\Psi, D) \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ D \supset_c A \Uparrow}\ \mathbf{cfrm}- \supset_c R$$

$$\frac{\Psi \models D \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ D \wedge_c A \Uparrow}\ \mathbf{cfrm} - \wedge_c R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Longrightarrow_0\ \ B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A \multimap B \Uparrow}\ \mathbf{cfrm}- \multimap R_v$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A\&B \Uparrow}\ \mathbf{cfrm} - \&R_{00}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega \Longrightarrow_1\ \ B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A\&B \Uparrow}\ \mathbf{cfrm} - \&R_{01}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_1;\ \Omega \Longrightarrow_1\ \ A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A\&B \Uparrow}\ \mathbf{cfrm} - \&R_{10}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O1};\ \Omega \Longrightarrow_1\ \ A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O2};\ \Omega \Longrightarrow_1\ \ B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega \Longrightarrow_1\ \ A\&B \Uparrow}\ \mathbf{cfrm} - \&R_{11}$$

$$\frac{\Psi \mid \Gamma\text{,A};\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ A \supset B \Uparrow}\ \mathbf{cfrm}- \supset R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ [a/x]A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \forall x.A \Uparrow}\ \mathbf{cfrm} - \forall R$$

$$\frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ \Omega \Longrightarrow_1\ \ \mathrm{T} \Uparrow}\ \mathbf{cfrm} - TR$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \ C, \quad C \textit{ not right asynchronous}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ C \Uparrow}\ \mathbf{cfrm}- \Uparrow R$$

Figure 4.1: CFRM, Right invertible rules

$$\frac{(\Psi, D) \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_v\ \mathrm{C}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, D \wedge_c A \Uparrow \Longrightarrow_v\ C}\ \mathbf{cfrm} - \wedge_c L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A, B \Uparrow \Longrightarrow_v\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \otimes B \Uparrow \Longrightarrow_v\ C}\ \mathbf{cfrm} - \otimes L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega, B \Uparrow \Longrightarrow_1\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ C}\ \mathbf{cfrm} - \oplus L_{01}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_1;\ \Omega, A \Uparrow \Longrightarrow_1\ C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_0\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ C}\ \mathbf{cfrm} - \oplus L_{10}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_0\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ C}\ \mathbf{cfrm} - \oplus L_{00}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O1};\ \Omega, A \Uparrow \Longrightarrow_1\ \mathrm{C} \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O2};\ \Omega, B \Uparrow \Longrightarrow_1\ \mathrm{C}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega, A \oplus B \Uparrow \Longrightarrow_1\ \mathrm{C}}\ \mathbf{cfrm} - \oplus L_{11}$$

$$\frac{\Psi \mid \Gamma,\mathrm{A};\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \mathrm{C}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega,!\mathrm{A} \Uparrow \Longrightarrow_v\ \mathrm{C}}\ \mathbf{cfrm}{-}!L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, [a/x]A \Uparrow \Longrightarrow_v\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, \exists x.A \Uparrow \Longrightarrow_v\ C}\ \mathbf{cfrm} - \exists L$$

$$\frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \cdot;\ \Omega,0 \Uparrow \Longrightarrow_0\ \mathrm{C}}\ \mathbf{cfrm} - 0L \qquad \frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \mathrm{C}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega,1 \Uparrow \Longrightarrow_v\ \mathrm{C}}\ \mathbf{cfrm} - 1L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I, A \setminus \Delta_O, A;\ \Omega \Uparrow \Longrightarrow_1\ C, \quad A\ not\ left\ asynchronous}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_1\ C}\ \mathbf{cfrm}{-} \Uparrow L_{1A}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I, A \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ C, \quad A\ not\ left\ asynchronous}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_v\ C}\ \mathbf{cfrm}{-} \Uparrow L_v$$

Figure 4.2: CFRM, Left invertible rules

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ C \Downarrow,\quad C \textit{ not atomic}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ C}\ \textit{decideR}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Psi \mid \Gamma;\ \Delta_I, A \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \mathrm{C}}\ \textit{decideL}$$

$$\frac{\Psi \mid \Gamma, A;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ \mathrm{C}}{\Psi \mid \Gamma, A;\ \Delta_I \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \mathrm{C}}\ \textit{decideL!}$$

Figure 4.3: *CFRM*, Decision rules

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_M;\ \cdot \Longrightarrow_v\ \mathrm{A} \Downarrow \quad \Psi \mid \Gamma;\ \Delta_M \setminus \Delta_O;\ \cdot \Longrightarrow_w\ \mathrm{B} \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_{v \vee w}\ \mathrm{A} \otimes \mathrm{B} \Downarrow}\ \mathbf{cfrm} - \otimes R$$

$$\frac{\Psi \mid \Gamma;\ \cdot \setminus \cdot;\ \cdot \Longrightarrow_v\ \mathrm{A} \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ \cdot \Longrightarrow_0\ !\mathrm{A} \Downarrow}\ \mathbf{cfrm}{-}!R \qquad \frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ \cdot \Longrightarrow_0\ 1 \Downarrow}\ \mathbf{cfrm} - 1R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \oplus \mathrm{B} \Downarrow}\ \mathbf{cfrm} - \oplus R_1$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{B} \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \oplus \mathrm{B} \Downarrow}\ \mathbf{cfrm} - \oplus R_2$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ [\mathrm{t/x}]\mathrm{A} \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \exists \mathrm{x.A} \Downarrow}\ \mathbf{cfrm} - \exists R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \Uparrow \quad \text{not right synchronous}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \mathrm{A} \Downarrow}\ \mathbf{cfrm}{-} \Downarrow R$$

Figure 4.4: CFRM, Right focusing rules

$$\frac{\Psi \models D \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Downarrow \Longrightarrow_v\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, D \supset_c A \Downarrow \Longrightarrow_v\ C}\ \mathbf{cfrm}{-}\supset_c L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_M;\ \cdot \Longrightarrow_v\ A \Downarrow \quad \Psi \mid \Gamma;\ \Delta_M \setminus \Delta_O;\ B \Downarrow \Longrightarrow_w\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \multimap B \Downarrow \Longrightarrow_{v \vee w}\ C}\ \mathbf{cfrm}{-}\multimap L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ B \Downarrow \Longrightarrow_v\ \mathrm{C} \quad \Psi \mid \Gamma;\ \cdot \setminus \cdot;\ \cdot \Longrightarrow_w\ A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \supset B \Downarrow \Longrightarrow_v\ \mathrm{C}}\ \mathbf{cfrm}{-}\supset L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \& B \Downarrow \Longrightarrow_v\ C}\ \mathbf{cfrm} - \&L_1$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ B \Downarrow \Longrightarrow_v\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \& B \Downarrow \Longrightarrow_v\ C}\ \&L_2$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ [t/x]A \Downarrow \Longrightarrow_v\ C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \forall x.A \Downarrow \Longrightarrow_v\ C}\ \mathbf{cfrm} - \forall L$$

$$\frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ P \Downarrow \Longrightarrow_0\ P}\ \mathbf{cfrm} - \mathit{init}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \Uparrow \Longrightarrow_v\ C \quad A\ \mathit{not\ atomic\ and\ not\ left\ synchronous}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ C}\ \mathbf{cfrm}{-}\Downarrow L$$

Figure 4.5: CFRM, Left focusing rules

In our case, we avoid these issues by restricting our domains to have only convex[1] constraints and no interpreted functions during unification. Below, we examine in depth these cases and propose solutions on how to handle them.

- **Case Splitting**

Numerical constraint satisfaction problems (*NCSP*) are defined as a set of constraints on variables. In *NCSP*, constraints are considered in conjunction and a solution is an assignment of values to the variables such that all the constraints are satisfied. For some problem domains, we may use constraints in disjunction form, which we call disjunctive numerical constraint satisfaction problems (*DNCSPs*).

As an example for *DNCSPs*, we define three constraints over the variable x:

$$C_1 = -3 < x < -1$$
$$C_2 = 0 < x < 3$$
$$C_3 = 2 < x < 4\ .$$

Afterwards, combining these constraints with disjunction such as $C_1 \vee C_2 \vee C_3$, we obtain a *DNCSP*. Considering another example, negation of an atom which is a numerical constraint in a logical formula can be expressed as another formula,

$$\neg(x = y) \longrightarrow (x < y) \vee (x > y).$$

Some work has been done on dealing with disjunctions of constraints. In [34], Ratschan proposes an extension of constraint programming (*CP*) framework to quantified first-order logical formulas whose atoms are numerical constraints. In [8], Douillard and Jermann introduce the concept of *interesting points* that represents potential splitting points and define some splitting heuristics for *DNCSPs*.

In our system, we don't handle case splitting. We believe that adding the following rule would handle possible nonconvexities in the constraint domain.

[1]We call a constraint formula non-convex if it entails a disjunction of constraints without entailing any of the constraints alone.

$$\frac{\Psi \models C_1 \vee C_2 \quad \Psi, C_1 \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P \quad \Psi, C_2 \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P} \; \textit{split}$$

The below example shows necessity of *split* rule for some cases. If we try to prove

$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P \Uparrow ,$$

where we have,

$\Psi : 1 < x < 4 \; (x \in \mathbb{N})$,
$\Gamma : a(x)$,
$\Delta_I : \cdot$,
$\Omega : \cdot$,
$P : ((a(2) \multimap a) \otimes (a(3) \multimap b)) \multimap (a \otimes b)$,

we should split the constraint in Ψ using *split* rule as:

$$\frac{\Psi \models (x = 2) \vee (x = 3) \quad \Psi, (x = 2) \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P \quad \Psi, (x = 3) \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P} \; \textit{split}$$

Although *CFRM* does not include *split* rule, SWI-Prolog implementation of *CFRM* solves the above example. For each time using the a(x) from unrestricted rules, x is replaced with a different variable. Thus, no contradiction case occurs while unification of a(x) with a(2) and a(3).

CFRM can handle above example, however, the example belows illustrates the necessity of case splitting even for *CFRM*. Considering the sequent as:

$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \; P \Uparrow ,$$

where we have,

$\Psi : 1 < x < 4 \ (x \in \mathbb{N})$,
$\Gamma : \exists x.a(x)$,
$\Delta_I : \cdot$,
$\Omega : \cdot$,
$P : a(2) \oplus a(3)$,

we should split the constraint in Ψ using *split* rule as:

$$\frac{\Psi \models (x = 2) \vee (x = 3) \quad \Psi, (x = 2) \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v \ P \quad \Psi, (x = 3) \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v \ P}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v \ P}\ \mathit{split}$$

Without this rule, entailment of $1 < x < 4 \models (x = 2)$ or $1 < x < 4 \models (x = 3)$ is not true. Since x may have values between 1 to 4, we can not assign any value to it. That is why *CFRM* can not prove above example.

- **Supporting for Interpreted Symbols During Unification**

Interpreted function symbols consist of arithmetic and logic operators which are built-in functions. In the initial sequent, if we have the init rule as

$$\frac{\Psi \models \mathbf{s} \doteq \mathbf{t}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ P(s) \Downarrow \Longrightarrow_v \ P(t)}\ \mathit{init}$$

where $\mathbf{s}$ and $\mathbf{t}$ are in vector form, we can go through the uninterpreted function symbols in all term arguments and isolate interpreted term equalities. Afterwards, we can transfer interpreted term equalities to the constraint domain.

Considering an example, $P(0-1)$ and $P(1-2)$ both include a proposition P and an interpreted function, because they have a built-in operator minus $(-)$. Since they evaluate to the same atomic proposition $P(-1)$, both of them are unified together.

In our implementation, we can implicitly use unification in the init rule as described here. Returning to the same example above, if we want to unify $P(0-1)$

with $P(1-2)$, we can add two constraints $X = 0 - 1$ and $Y = 1 - 2$ into the constraint store, and unify $P(X)$ with $P(Y)$. So, X is unified with Y.

There is still another problem which the modification of the init rule as above can not solve. The constraint solver is designed such that all variables in a constraint are implicitly in the scope of for-all ($\forall$) quantifiers. The constraint store Ψ may contain constraints with existential variables. In that case, insufficient information of existential variable causes failure of constraint entailment. We give an example for this case where Y and X are existential variables. So, X and Y both should be treated as an existential variable:

$$
\begin{gathered}
If\ \Psi\ =\ (Y = X - 1)\ , \\
then\ \Psi \models (Y = -1)\ fails\ , \\
while \\
If\ \Psi\ =\ (Y = 0 - 1)\ , \\
then\ \Psi \models (Y = -1)\ is\ true\ .
\end{gathered}
$$

Thus, we add each constraint into the constraint store instead of checking entailment. Considering the same example above, first case does not fail anymore.

$$
\begin{gathered}
If\ \Psi\ =\ (Y = X - 1)\ , \\
\{\Psi,\ (Y = -1)\}\ is\ true\ .
\end{gathered}
$$

However, this modification may disrupt the soundness property of *CFRM* system.

Adding constraint rules into *FRM* resulting more powerful system *CFRM*. Thus, we can handle both discrete and continues properties of robotic behaviors. Using constraint solvers of SWI-Prolog helps us reducing the complexity associated with the encoding. Nevertheless, there are some restrictions with constraints such as case splitting and support for interpreted symbols during unification. However, we can handle the latter restriction with some encoding modifications of constraints.

4.3 Annotated Proof Terms for CFRM

We introduced a new theorem prover system, *CFRM*, in Chapter 4.2. This system answers either *yes* if a given theorem is provable or *no* if a given theorem is not provable. We can also use this system for robotic planning such as a given goal is achievable or not in a defined domain. However, *CFRM* does not give any information for the constructed plan. We want to carry this work one step further and we want to record the constructed proof to yield corresponding plan to be used in our domain. To this end, we enrich *CFRM* with proof terms that carry enough information to reconstruct deduction of the proof. We achieve this with a notation for derivations to be carried along in deductions.

4.3.1 Grammar of Proof Terms

In this section, we describe the proof term grammar below.

Proof terms	I	::=	u	(variables)
			$\mid I \otimes I \mid \mathbf{let}_{\otimes}\ u_1 \otimes u_2 = I\ \mathbf{in}\ I$	$(A \otimes B)$
			$\mid \star \mid \mathbf{let}_1\ \star = I\ \mathbf{in}\ I$	(1)
			$\mid \langle I, I \rangle \mid fst\ I \mid snd\ I$	$(A \& B)$
			$\mid \langle\rangle$	(T)
			$\mid \langle u, I \rangle_{\exists} \mid \mathbf{let}_{\exists}\ \langle u_1, u_2 \rangle_{\exists} = I\ \mathbf{in}\ I$	$(\exists x.A)$
			$\mid inl^C I \mid inr^C I \mid (\mathbf{case}\ I\ \mathbf{of}\ inl\ u_1 \Rightarrow I_1 \mid inr\ u_2 \Rightarrow I_2)$	$(A \oplus B)$
			$\mid \lambda u.I \mid II$	$(A \multimap B)$
			$\mid \hat{\lambda} u.I \mid I \hat{} I$	$(A \supset B)$
			$\mid \lambda^{\forall} u.I \mid I^{\forall} I$	$(\forall .xA)$
			$\mid abort^C\ I$	(0)
			$\mid !I \mid \mathbf{let}_{bang}\ !u = I\ \mathbf{in}\ I$	$(!A)$
			$\mid \lambda^c\{D\}.\ I \mid I\{D\}$	$(D \supset_c A)$
			$\mid \{D, I\} \mid \mathbf{let}_c\{D, I\} = I\ \mathbf{in}\ I$	$(D \wedge_c A)$

Our proof term grammar is mostly inspired from similar existing formulations in the literature [31, 32]. However, we also incorporate constraint expressions. We do not explicitly record constraint proofs since we trust that the constraint solver can recheck the constraints at the program execution. In the proof term grammar, D denotes constraint expressions.

4.3.2 Rule Set of Proof Terms

In this section, first we describe judgments of proof terms. Afterwards, we give rule set of proof terms. The proof term assignment is defined via four judgments:

$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_v \;\; I : C \Uparrow ,$$
$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Uparrow \Longrightarrow_v \;\; I : C ,$$
$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \cdot \Longrightarrow_v \;\; I : C \Downarrow ,$$
$$\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \Downarrow \Longrightarrow_v \;\; I : C ,$$

where we have,

Ψ	:	Constraint store,
Γ	:	Unrestricted hypotheses ,
Δ_I	:	Input resources that may be consumed ,
Δ_O	:	Output resources that are not consumed ,
Ω	:	Ordered hypotheses ,
C	:	The goal ,
A	:	The focused proposition ,
v	:	Flag to encode freedom in resource consumption (0 or 1) ,
I	:	Proof Term ,
u	:	Label ,

and each formula in Γ , Δ_I , Δ_O and Ω is labeled. Rule set of proof terms correspond with the annotations is given in Figure 4.6, 4.7, 4.8, 4.9 and 4.10.

We give some examples, yielding proof extraction of *CFRM* system. The following proof objects are generated by our theorem prover.

$$\begin{aligned}
\lambda v_1.\lambda v_1 &: a \multimap a \\
\lambda v_1.\mathbf{let}\ v_2 \otimes v_3 = v_1\ \mathbf{in}\ v_3 \otimes v_2 &: (b \otimes a) \multimap (a \otimes b) \\
\lambda v_1.(\mathbf{case}\ v_1\ of\ \mathbf{inl}\ v_2 \Rightarrow v_2 \mid \mathbf{inr}\ v_3 \Rightarrow v_3) &: (a \oplus a) \multimap a \\
\lambda v_1.\lambda v_2.((v_2)^*(1))(v_1) &: a(1) \multimap (\forall x.a(x) \multimap b(x)) \multimap b(1)
\end{aligned}$$

$$\frac{(\Psi, D) \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ I : A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \lambda\{D\}.\, I : D \supset_c A \Uparrow}\ \mathbf{cfrm}- \supset_c R$$

$$\frac{\Psi \models D \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ I : A \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \{D, I\} : D \wedge_c A \Downarrow}\ \mathbf{cfrm} - \wedge_c R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A \Longrightarrow_v\ \ I : B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \hat{\lambda} u.I : A \multimap B \Uparrow}\ \mathbf{cfrm}- \multimap R_v$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ I : A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ J : B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ \langle I, J\rangle : A \& B \Uparrow}\ \mathbf{cfrm} - \&R_{00}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ I : A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega \Longrightarrow_1\ \ \mathrm{J} : \mathrm{B} \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ \langle I, J\rangle : A \& B \Uparrow}\ \mathbf{cfrm} - \&R_{01}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_1;\ \Omega \Longrightarrow_1\ \ I : A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ \mathrm{J} : \mathrm{B} \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ \langle I, J\rangle : A \& B \Uparrow}\ \mathbf{cfrm} - \&R_{10}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O_1};\ \Omega \Longrightarrow_1\ \ I : A \Uparrow \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O_2};\ \Omega \Longrightarrow_1\ \ \mathrm{J} : \mathrm{B} \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O_1} \cap \Delta_{O_2};\ \Omega \Longrightarrow_1\ \ \langle I, J\rangle : A \& B \Uparrow}\ \mathbf{cfrm} - \&R_{11}$$

$$\frac{\Psi \mid \Gamma, \mathrm{u{:}A};\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ I : B \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \lambda u.I : A \supset B \Uparrow}\ \mathbf{cfrm}- \supset R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ I : [a/x]A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \lambda^{\forall} x.[x/a]I : \forall x.A \Uparrow}\ \mathbf{cfrm} - \forall R$$

$$\frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ \Omega \Longrightarrow_1\ \ \langle\,\rangle : T \Uparrow}\ \mathbf{cfrm} - TR$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \ I : C, \quad C \textit{ not right asynchronous}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ I : C \Uparrow}\ \mathbf{cfrm}- \Uparrow R$$

Figure 4.6: Proof terms for *CFRM*, Right invertible rules

$$\frac{(\Psi, D) \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, w : A \Uparrow \Longrightarrow_v \quad I : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : D \wedge_c A \Uparrow \Longrightarrow_v \quad \mathbf{let}\ \{D, w\} = u\ \mathbf{in}\ I : C}\ \mathbf{cfrm} - \wedge_c L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A, w : B \Uparrow \Longrightarrow_v \quad I : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, k : A \otimes B \Uparrow \Longrightarrow_v \quad \mathbf{let}\ u \otimes w = k\ \mathbf{in}\ I : C}\ \mathbf{cfrm} - \otimes L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, v : A \Uparrow \Longrightarrow_0 \quad I : C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega, w : B \Uparrow \Longrightarrow_1 \quad J : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A \oplus B \Uparrow \Longrightarrow_0 \quad (\mathbf{case}\ u\ \mathbf{of}\ inlv \Rightarrow I \mid inrw \Rightarrow J) : C}\ \mathbf{cfrm} - \oplus L_{01}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_1;\ \Omega, v : A \Uparrow \Longrightarrow_1 \quad I : C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, w : B \Uparrow \Longrightarrow_0 \quad J : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A \oplus B \Uparrow \Longrightarrow_0 \quad (\mathbf{case}\ u\ \mathbf{of}\ inlv \Rightarrow I \mid inrw \Rightarrow J) : C}\ \mathbf{cfrm} - \oplus L_{10}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, v : A \Uparrow \Longrightarrow_0 \quad I : C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, w : B \Uparrow \Longrightarrow_0 \quad J : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A \oplus B \Uparrow \Longrightarrow_0 \quad (\mathbf{case}\ u\ \mathbf{of}\ inlv \Rightarrow I \mid inrw \Rightarrow J) : C}\ \mathbf{cfrm} - \oplus L_{00}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O_1};\ \Omega, v : A \Uparrow \Longrightarrow_1 \quad I : C \quad \Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O_2};\ \Omega, w : B \Uparrow \Longrightarrow_1 \quad J : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega, u : A \oplus B \Uparrow \Longrightarrow_1 \quad (\mathbf{case}\ u\ \mathbf{of}\ inlv \Rightarrow I \mid inrw \Rightarrow J) : C}\ \mathbf{cfrm} - \oplus L_{11}$$

$$\frac{\Psi \mid \Gamma, \mathrm{u{:}A};\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v \quad I : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, w : !A \Uparrow \Longrightarrow_v \quad \mathbf{let}\ !u = w\ \mathbf{in}\ I : C}\ \mathbf{cfrm}{-}!L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, w : [a/x]A \Uparrow \Longrightarrow_v \quad I : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : \exists x.A \Uparrow \Longrightarrow_v \quad \mathbf{let}\ \langle x, w\rangle_\exists = u\ \mathbf{in}\ [x/a]I : C}\ \mathbf{cfrm} - \exists L$$

$$\frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \cdot;\ \Omega, u : 0 \Uparrow \Longrightarrow_0 \quad abort^C\ u : C}\ \mathbf{cfrm} - 0L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v \quad I : C}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : 1 \Uparrow \Longrightarrow_v \quad \mathbf{let}\ \star = u\ \mathbf{in}\ I : C}\ \mathbf{cfrm} - 1L$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I, u : A \setminus \Delta_O, u : A;\ \Omega \Uparrow \Longrightarrow_1 \quad I : C, \quad A\ not\ left\ asynchronous}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A \Uparrow \Longrightarrow_1 \quad I : C}\ \mathbf{cfrm}{-} \Uparrow L_{1A}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I, u : A \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v \quad I : C, \quad A\ not\ left\ asynchronous}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, u : A \Uparrow \Longrightarrow_v \quad I : C}\ \mathbf{cfrm}{-} \Uparrow L_v$$

Figure 4.7: Proof terms for *CFRM*, Left invertible rules

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ I : C \Downarrow,\quad C \textit{ not atomic}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \ I : C}\ \mathbf{cfrm} - \textit{decideR}$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ u : A \Downarrow \Longrightarrow_v\ \ I : C}{\Psi \mid \Gamma;\ \Delta_I, u : A \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \ I : C}\ \mathbf{cfrm} - \textit{decideL}$$

$$\frac{\Psi \mid \Gamma, u : A;\ \Delta_I \setminus \Delta_O;\ u : A \Downarrow \Longrightarrow_v\ \ I : C}{\Psi \mid \Gamma, u : A;\ \Delta_I \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ \ I : C}\ \mathbf{cfrm} - \textit{decideL}!$$

Figure 4.8: Proof terms for *CFRM*, Decision rules

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_M;\ \cdot \Longrightarrow_v\ \ I : A \Downarrow \quad \Psi \mid \Gamma;\ \Delta_M \setminus \Delta_O;\ \cdot \Longrightarrow_w\ \ J : B \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_{v \vee w}\ \ I \otimes J : A \otimes B \Downarrow}\ \mathbf{cfrm} - \otimes R$$

$$\frac{\Psi \mid \Gamma;\ \cdot \setminus \cdot;\ \cdot \Longrightarrow_v\ \ I : A \Uparrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ \cdot \Longrightarrow_0\ \ !I : !A \Downarrow}\ \mathbf{cfrm}{-}!R$$

$$\frac{}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_I;\ \cdot \Longrightarrow_0\ \ \star : 1 \Downarrow}\ \mathbf{cfrm} - 1R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ I : A \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ inl^B I : A \oplus B \Downarrow}\ \mathbf{cfrm} - \oplus R_1$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ I : B \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ inr^A I : A \oplus B \Downarrow}\ \mathbf{cfrm} - \oplus R_2$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ I : [t/x]A \Downarrow}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ \langle t, I\rangle : \exists x.A \Downarrow}\ \mathbf{cfrm} - \exists R$$

$$\frac{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ I : A \Uparrow \textit{ not right synchronous}}{\Psi \mid \Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ I : A \Downarrow}\ \mathbf{cfrm}{-} \Downarrow R$$

Figure 4.9: Proof terms for *CFRM*, Right focusing rules

$$\frac{\Psi \models D \quad \Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega, w : A \Downarrow \Longrightarrow_v \;\; I : C}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega, u : D \supset_c A \Downarrow \Longrightarrow_v \;\; [u\{D\}/w]\; I : C}\; \mathbf{cfrm}{-}\supset_c L$$

$$\frac{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_M;\, \cdot \Longrightarrow_v \;\; J : A \Downarrow \quad \Psi \mid \Gamma;\, \Delta_M \setminus \Delta_O;\, w : B \Downarrow \Longrightarrow_k \;\; I : C}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \multimap B \Downarrow \Longrightarrow_{v \vee k} \;\; [\hat{u}J/w]\; I : C}\; \mathbf{cfrm}{-}\multimap L$$

$$\frac{\Psi \mid \Gamma;\, \cdot \setminus \cdot;\, \cdot \Longrightarrow_v \;\; J : A \Uparrow \quad \Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, w : B \Downarrow \Longrightarrow_k \;\; I : C}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \supset B \Downarrow \Longrightarrow_k \;\; [uJ/w]I : C}\; \mathbf{cfrm}{-}\supset L$$

$$\frac{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, w : A \Downarrow \Longrightarrow_v \;\; I : C}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \& B \Downarrow \Longrightarrow_v \;\; [fst\ u/w]I : C}\; \mathbf{cfrm} - \&L_1$$

$$\frac{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, w : B \Downarrow \Longrightarrow_v \;\; I : C}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \& B \Downarrow \Longrightarrow_v \;\; [snd\ u/w]I : C}\; \mathbf{cfrm} - \&L_2$$

$$\frac{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, w : [t/x]A \Downarrow \Longrightarrow_v \;\; I : C}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : \forall x.A \Downarrow \Longrightarrow_v \;\; [u^{\forall}t/w]I : C}\; \mathbf{cfrm} - \forall L$$

$$\frac{}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_I;\, u : P \Downarrow \Longrightarrow_0 \;\; u : P}\; \mathbf{cfrm} - init$$

$$\frac{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \Uparrow \Longrightarrow_v \;\; I : C \quad A\ not\ atomic\ and\ not\ left\ synchronous}{\Psi \mid \Gamma;\, \Delta_I \setminus \Delta_O;\, u : A \Downarrow \Longrightarrow_v \;\; I : C}\; \mathbf{cfrm}{-}\Downarrow L$$

Figure 4.10: Proof terms for *CFRM*, Left focusing rules

In this chapter, we described *CILL* system which is generated by adding constraints into *Intuitionistic Linear Logic*. Afterwards, we incorporated constraints into *FRM* system, yielding *CFRM* system. We prove neither the soundness nor the completeness of *CFRM* system, since it is nontrivial to prove. We also include some restrictions with constraints into this chapter such as case splitting and support for interpreted symbols during unification. At the last section of this chapter, we introduced annotated proof terms for *CFRM* system, helping proof extraction.

Chapter 5

Implementation Details and Experiment Results

In this chapter, we will give a detailed information about the implementation environment, the implementation of *FRM* and *CFRM* systems, and also some planning examples for *CFRM* system.

5.1 SWI-Prolog as the Programming Environment

Prolog is a declarative programming language. In the declarative programming languages, there are definitions and statements in programs to achieve some goals. When a computer executes a logic program, it uses the inference rules which are given in the program to derive the derived rules. The main purpose is to find a solution for the given goal. Prolog tries to solve the goals by searching all possible ways which are given in the program. For more information about Prolog, see [2, 29]. To implement our proof system *FRM*, we use SWI-Prolog[1] which is an open source implementation of the programming language Prolog. The

[1]Home page is: http://www.swi-prolog.org

main reason using this prolog environment is that it provides some libraries such as *CHR* (Constraint Handling Rules), clpfd (over Finite Domains), clpqr(over Rationals and Reals) and various others, for the constraint logic programming.

We use "clpr" (constraint logic programming over reals) library among other libraries for handling constraints over the real numbers (using floating point numbers as representation).

We use some predicates in SWI-Prolog for handling constraints in the implementation. We give short descriptions about these predicates.

- **{}***(+Constraints)*:

  Adds the constraints given by Constraints to the constraint store.

- **entailed**(*+Constraint*):

  Succeeds if Constraint is necessarily true within the current constraint store. This means that adding the negation of the constraint to the store results in failure.

Using the unification mechanism is also another option to store constraints, instead of using the { }/1 predicate. The following code samples are equivalent:

- Unification with a variable

  {X =:= Y}

  {X = Y}

  X = Y

- Unification with a number

  {X =:= 5.0}

  {X = 5.0}

  X = 5.0

5.2 Implementation of FRM

In SWI-Prolog implementation of *FRM* system, there are four main sub-predicates which are right_invertible_rules, left_invertible_rules, right_focusing and left_focusing. The main predicate is *prove* and our theorem prover is invoked by this predicate. One can use *prove* predicate with two different variations. First one is invoked as *prove(Unrestricted Hypotheses, Resources, Goal)*. Second variation is *prove(Unrestricted Hypotheses, Resources, Goal, Actions)* which has one more parameter, *Actions*. At the end of the run, if there is any solution for the goal, applied actions during the search are returned in the *Actions* context. Operators and their equivalents used in the implementation are on the below:

$$\text{linImp}(\multimap),\ \Rightarrow(\supset),\ +(\oplus),\ \text{x}(\otimes),\ \&,\ \text{all}(\forall),\ \text{ex}(\exists).$$

5.2.1 Skolemization in FRM

For the right rule of existential($\exists$) quantifier and the left rule of for all($\forall$) quantifier, a new term t is introduced in $\exists x.A$ or $\forall x.A$, and all occurrences of variable x in A is replaced by this new term. However, we need to replace x with a new variable a in A for the left rule of $\exists$ and for the right rule of $\forall$. Thus, this new variable must be a unique variable which is not defined before and this variable must not be unified with other variables or terms except itself during unification process. Therefore, we replace the quantified variable x by a Skolem function which is created as `S^ FV` where S is the level of the formula tree, and FV is the free variable list. If we call again the example $\exists y.\forall x.y = x$;

$$\begin{aligned} &\exists y.\forall x.y = x \\ &\forall x.Y = x \qquad (\exists I) \\ &Y = S\hat{}[Y] \qquad (\forall I) \\ &\# \end{aligned}$$

5.2.2 Unification in FRM

In 2.4.2, we mentioned unification, however unification in the implementation deserves some more explanations. The atomic propositions are handled and unified in the init rule. In *FRM* system, right invertible, left invertible, right focusing and left focusing rules are applied, respectively. Init rule is in the end of left focusing rule set, thus init rule is applied last. In some conditions, it would increase the performance of the system if we could unify the atomic propositions whenever an atomic proposition is first encountered on the right. As an example, assume that the initial case of a sequent is given as

$$\Gamma; \Delta_I \setminus \Delta_O; \Omega, P \Longrightarrow P$$

where Ω is not empty (at least there is a left invertible term) and P is an atomic proposition. In our *FRM* system, first right invertible rules are tried to apply. Since P is an atomic proposition and is not considered as a right invertible term, non of the rules are applicable. So next trial is done over left invertible rules. All left invertible rules are applied until none of left invertible terms are left. Since in the above example, Ω has at least one left invertible term, the appropriate rule is applied. Unification of P is only possible in the init rule, where there is a P on the both sides. But somehow, if we can recognize that there is an atomic term on the right hand side during application of right invertible rules, we can search a matching of atomic term in Ω. In this example, P is also on the left hand side, thus, we can catch a matching. This will reduce the search space and as a conclusion, the performance will be increased.

Standard Unification Algorithm

In Prolog, standard unification is done according to the following algorithm:

1. If T1 and T2 are constants, then T1 and T2 unify if they are the same atom, or the same number.

2. If T1 is a variable and T2 is any type of term, then T1 and T2 unify, and T1 is instantiated to T2. (and vice versa)

3. If T1 and T2 are complex terms, then they unify if:

   (a) They have the same functor and arity, and

   (b) all their corresponding arguments unify, and

   (c) the variable instantiations are compatible.

In some cases, the standard prolog unification does not work. For an instance, if we run in Prolog

?- A = f(A).

we get

A = f(f(f(f(f(f(f(f(f(f(...))))))))))

as a result. To solve infinite condition, we use a special predicate, *unify_with_occurs_check*. If we run in Prolog

?- unify_with_occurs_check(A, f(A)).

we get

No

meaning that unification fails.

Herbrand's Unification Algorithm

We give *Herbrand Algorithm* [17] for the most general unifier (MGU) which is more comprehensive than the *Standard Unification Algorithm*.

Given a set of equations of the form $t1 = t2$, apply in any order one of the following non-exclusive steps:

1. If there is an equation of the form:

   (a) f=g where f and g are different atomic terms, or

   (b) f=g where f is an atomic term and g is a compound term, or f is a compound term and g is an atomic term, or

   (c) f(...) = g(...) where f and g are different functors, or

   (d) f(a1, a2, ..., aN) = g(b1, b2, ..., bM) where N and M are different,

   then exit with failure (not unifiable).

2. If there is an equation of the form X = X, X being a variable, then remove it.

3. If there is an equation of the form c = c, c being a atomic term, then remove it.

4. If there is an equation of the form f(a1, a2, ..., aN) = f(b1, b2, ..., bN) then replace it by the set of equations ai = bi.

5. If there is an equation of the form t = X, X being a variable and t a non-variable term, then replace it by the equation X = t,

6. If there is an equation of the form X = t where:

   (a) X is a variable and t is a term in which the variable does not occur, and

   (b) the variable X occurs in some other equation, then substitute in all other equations every occurrence of the variable X by the term t.

7. If there is an equation of the form X = t such that X is a variable and t is a non-variable term which contains this variable, then exit with failure (not unifiable, positive occurs check.

8. If no other step is applicable, then exit with success (unifiable).

5.2.3 Depth-Limited Depth First Search in FRM

We use *Depth-Limited Depth-First Search (DLDFS)* method for the proof search in *FRM* system. Before giving details about *DLDFS*, we give the definition of *Depth-First Search (DFS)* algorithm since *DLDFS* is derived from *DFS*. *DFS* is an algorithm that always expands one of the nodes at the deepest level of the tree, until finds a goal node [37]. If the search hits a dead end which means a nongoal node with no expansion, then the search backtracks and expand nodes at shallower levels. The drawback of *DFS* is that it can get stuck going down the wrong path (an infinite loop occurs), even when a shallow solution exists. The solution is imposing a cutoff on the maximum depth of a path for *DFS* which we call this method *Depth-Limited Depth-First Search (DLDFS)*. The basic idea about *DLPFS* is that we do not only apply *DFS* but also give a depth limit which determines number of level to continue for searching. Considering when we try to prove the sequent $((p \oplus (p \supset \bot)) \supset \bot) \supset \bot$ with *DFS* method, applying *FRM* rule set to this sequent will cause an infinite loop. However, when we limit the depth with an adequate number, proof will be completed. In the implementation, there is an upper limit. Proof search starts with an initial depth limit, 1, increasing it one by one for each failure until reaching upper depth limit.

5.3 Implementation of CFRM

We divided SWI-Prolog code of *CFRM* system into subroutines. Two variations of *prove* predicate is used for *FRM* system. The last variation has one more parameter then the previous ones, *prove(Constraints, Unrestricted Hypotheses, Resources, Goal, Actions)*. This extra context *Constraints* holds constraints if the user wants to give some constraints at the beginning of the proof search. In addition to *FRM* rules, four more rules about constraints, $\supset_c R$, $\wedge_c R$, $\supset_c L$, $\wedge_c L$, are added for *CFRM* system.

5.3.1 Skolem Variables with Constraints

Since we use the skolem variables for $\forall R$ and $\exists L$ rules in the form of $L\hat{}\ F$, binding constraint variables with skolem variables is not possible. However, we can handle this case with a trivial trick. For each skolem variable, a new variable is created and these new variables are replaced with skolem variables. Shortly before the unification, if the variable is in the constraint store and also mapping a skolem variable, then we do not replace anything. However, if the variable is not in the constraint store but mapping a skolem variable, then we replace the variable with the skolem variable.

The following example is about above condition.

$$(\exists y.((y = 0 - 1) \wedge_c \mathsf{p}(y))) \multimap \mathsf{p}(-1)$$

The corresponding proof tree for this example is:

$$\dfrac{\dfrac{\dfrac{(a = 0 - 1) \mid \mathsf{p}(a) \Longrightarrow \mathsf{p}(-1)}{(a = 0 - 1) \wedge_c \mathsf{p}(a) \Longrightarrow \mathsf{p}(-1)} \wedge_c L}{\exists y.((y = 0 - 1) \wedge_c \mathsf{p}(y)) \Longrightarrow \mathsf{p}(-1)} \exists L^a}{\cdot \Longrightarrow (\exists y.((y = 0 - 1) \wedge_c \mathsf{p}(y))) \multimap \mathsf{p}(-1)} \multimap R$$

A parametric variable is not unified with a constant during unification. Considering unification of propositions $p(a)$ and $p(-1)$, for this reason they are not unified. However, having a constraint over the parametric variable helps unification. Since the parameter a is in the constraint store, a new variable mapping a is used in unification. $p(A) \Longrightarrow p(-1)$ is unified, assuming that A maps the parameter a and coherent with the constraint store.

We must also note that, the constraint solver is designed such that all variables in a constraint are implicitly in the scope of for-all($\forall$) quantifier. Hence, we can not use $\exists L$ rule and $\forall R$ rule within constraints.

5.4 Experiment Results

In this section, first we describe some theorems and give output of our implementation for these theorems. Afterwards, we describe some nontrivial planning problems to denote expressivity of *CFRM* system and discuss efficiency and performance of the system. Since we introduce new problem domains, non of the existing theorem provers have encodings and solutions of these problems. Thus, we can not compare the performance of our theorem prover with other provers. We must also note that, to be able to load the program in SWI-Prolog, one must type *['CIFOLL.pl']* on command line. This will load the program into the memory.

5.4.1 Program Outputs of Some Examples

In this part, we give SWI-Prolog outputs while proving some theorems. Program is called with the predicate *prove(A,B,C)* where

A : Unrestricted hypotheses,
B : Consumable resources,
C : The goal.

Representation of all operators used in the program and their equivalent operators are described below.

$linImp$: $\multimap$,
$=>$: $\supset$,
$impC$: $\supset_c$,
$andC$: $\wedge_c$,
$+$: $\oplus$,
x : $\otimes$,
$\&$: $\&$,
all : $\forall$,
ex : $\exists$.

We give a list of examples. First lines for each proof are program inputs and other lines are program outputs. Proof terms for each proof also can be seen in the outputs. We also note that the counter denotes the number of entered predicates and depth limit is given 50.

?- prove([ ], [b x a], (a x b)).
Program starts..
(let_x (v2 x v3) = v1 in (v3 x v2))
Total time is: 0.0
Counter is: 18
true

?- prove([ ],[ ],((a linImp (a1 & a2)) linImp (a linImp a1))).
Program starts..
(\ \ ^ v1. (\ \ ^ v2. (fst (app^ (v1, v2)))))
Total time is: 0.0
Counter is: 17
true

?- prove([ ], [a linImp b linImp c], b linImp a linImp c).
Program starts..
(\ \ ^ v2. (\ \ ^ v3. (app^((app^(v1, v3)), v2))))
Total time is: 0.0
Counter is: 26
true

?- prove([ ],[ ],((p + (p ⇒ 0)) ⇒ 0)⇒ 0).
Program starts..
(\\v116. (app(v116, (inr (\\v318. (app(v116, (inl v318))))))))
Total time is: 0.03
Counter is: 2584
true

?- prove([], [a + b, c], c).
Program starts..

false.

?- prove([ ], [ ], all X: (ex Y: ((all Y: p(Y)) ⇒ p(X)))).
Program starts..
(\\all-_G364. (pair_ex < _G364, (\\v2. (app_all(v2, _G364))) >))
Total time is: 0.0
Counter is: 23
true

?- prove([test act (all X: (at(X) linImp (ex Y: ((Y = X - 1) andC at(Y)))))],
[at(0)], at(-1.0)).
Program starts..
(let_ex (pair_ex< [app_lin, [app_all, [var, v2], 0], [var, v1]], v5 >) =
(appˆ((app_all(v2, 0)), v1)) in (let_c (pair_c-1.0=0-1, v6) = v5 in v6))
Total time is: 0.0
Counter is: 30
true

5.4.2 Blocks World Example

We recall the blocks world example from 2.2.4. The initial state of the environment is described as:

$$\Delta_0 = (empty,\ tb(a),\ tb(b),\ clear(a),\ clear(c)\ ,\ on(c,b)).$$

We define the goal as $tb(c)$. That means the block c will be on the table. The program input and output are described below:

We give rules, resources and the goal to the program as an input:

?- Grab = [
grab_on_block act (all X: (all Y:((empty x clear(X) x on(X,Y)) linImp (holds(X)
x clear(Y))))),
grab_on_table act (all X: ((empty x clear(X) x tb(X)) linImp holds(X))),

put_on_block act (all X: (all Y: ((holds(X) x clear(Y)) linImp (empty x clear(X) x on(X,Y))))),
put_on_table act (all X: (holds(X) linImp (empty x clear(X) x tb(X))))
],
Area = [empty, tb(a), tb(b), clear(a), clear(c), on(c,b)],
Goal = (tb(c) x erase),
prove(Grab, Area, Goal, Actions).

Proof terms are:

Program starts..
(let_x (v251 x v252) = (appˆ((app_all((app_all(v247, c)), b)), (v1 x (v5 x v6)))) in (let_x (v293 x v294) = (appˆ((app_all(v290, c)), v251)) in (let_x (v295 x v296) = v294 in (v296 x < >))))

The output of time, entered predicates and actions are given below:

Total time is: 0.02
Counter is: 2258
Grab = [grab_on_block act (all X:all Y: (empty x clear(...)x on(..., ...)linImp holds(X)x clear(Y))), grab_on_table act (all X: (empty x clear(X)x tb(X)linImp holds(X))), put_on_block act (all X:all Y: (...x...linImp...x...)), put_on_table act (all X: (holds(X)linImp empty x ...x...))],
Area = [empty, tb(a), tb(b), clear(a), clear(c), on(c, b)],
Goal = tb(c)x erase,
Actions = [put_on_table, grab_on_block]

In the action context, we can see actions used in planning. The robot arm first grabs the block c and later puts it on the table. This is a simple example but important to point the usage of *CFRM* system in planning domains.

5.4.3 Path Finding Among Mines

First of all, we define the domain of the planning example and give the initial states. Planning domain comprises an area (3x3) and a mobile robot which can move one cell for each time. Robot has a power constraint such that if there is not enough power to reach the goal, robot can not achieve the goal. Planning domain also includes mines on the area. If there is a mine on any point, the robot can not move to that point. The robot tries to find a mine free path from initial state to the goal state. We define three propositions, *at*, *free* and *power*. The proposition *at(x,y)* gives the location information about the robot and the proposition free(x,y) gives the mine free location. The proposition *power(p)* denotes the power of the robot. For instance, if the initial power is 5, the robot can not move more than 5 steps.

As an example, the initial state is illustrated in Figure 5.1. The robot is at the (x_1 ,y_1) position and there are mines at (x_2, y_1), (x_3, y_1), (x_1, y_3), (x_2, y_3), (x_3, y_3) points. Initial power of the robot is 3, which means that robot can move at most 3 steps. Reaching the (x_3, y_2) point is the goal in this example.

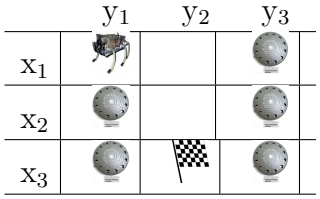

Figure 5.1: Robot can reach to the position x_3, y_2

For this example, below we encode resources and the goal for the program input. Four unrestricted rules (actions) are:

$\Gamma = [$
move down : $\forall x.\forall y.\forall z.(((z = x + 1), (p_1 = p - 1), (p_1 >= 0)) \wedge_c ((at(x, y) \otimes free(z, y) \otimes power(p)) \multimap (at(z, y) \otimes free(x, y) \otimes power(p_1))))$,
move right : $\forall x.\forall y.\forall z.(((z = y + 1), (p_1 = p - 1), (p_1 >= 0)) \wedge_c ((at(x, y) \otimes free(x, z) \otimes power(p)) \multimap (at(x, z) \otimes free(x, y) \otimes power(p_1))))$,

move up : $\forall x.\forall y.\forall z.(((z = x-1),(p_1 = p-1),(p_1 >= 0)) \wedge_c ((at(x,y) \otimes free(z,y) \otimes power(p)) \multimap (at(z,y) \otimes free(x,y) \otimes power(p_1))))$,
move left : $\forall x.\forall y.\forall z.(((z = y-1),(p_1 = p-1),(p_1 >= 0)) \wedge_c ((at(x,y) \otimes free(x,z) \otimes power(p)) \multimap (at(x,z) \otimes free(x,y) \otimes power(p_1))))$
].

Initial resources for the example are:

$$\Delta_0 = [power(3), at(1,1), free(1,2), free(2,2), free(3,2)].$$

And finally, the goal is:

$$G = at(3,2) \otimes T.$$

Constraint store (Ψ) is initially empty since at the beginning we do not define any constraints. But during the plan search, Ψ will have some constraints which come from actions. Using Γ, Δ_0 and G, our planner finds a path constructed from actions, [move right, move down, move down].

If we change the initial power from 3 to 2, the resource store will be:

$$\Delta_0 = [power(2), at(1,1), free(1,2), free(2,2), free(3,2)] \ .$$

With the same Γ, the planner can not find a path for the same goal G, since reaching the goal position is achieved at least 3 steps.

Considering the same initial resources and the goal above, adding one more mine at (x_2, y_2) as in Figure 5.2 results a failure for the robot achieving the goal.

We must indicate that, we can not define negation of any resource in the planning domain. Considering the above examples, we define free points with *free* proposition rather than *not_mine* proposition, telling there is not any mine. This restriction is caused by nature of the linear logic resources, since we can not show absence of a resource unlike in the classical logic.

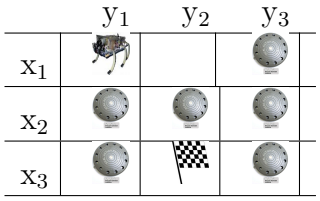

Figure 5.2: Robot can not reach to the position x_3, y_2

For now, we do not interpret proof terms which are complicated and a huge collection of terms, however, as a future work we plan to interpret proof terms and extract plans after the proof construction. For this example, the elapsed time is about some seconds for finding a proof. The input and output of the program are below.

Rules, resources and the goal are given to the program:

?- Rules = [
move_down act (all X: (all Y: (all Z: (all P: (all P1: (((Z = X + 1), (P1 = P-1), (P1 >= 0)) andC ((at(X,Y) x free(Z,Y) x power(P)) linImp (at(Z,Y) x free(X,Y) x power(P1))))))))),
move_right act (all X: (all Y: (all Z: (all P: (all P1: (((Z = Y + 1), (P1 = P-1), (P1 >= 0)) andC ((at(X,Y) x free(X,Z) x power(P)) linImp (at(X,Z) x free(X,Y) x power(P1))))))))),
move_up act (all X: (all Y: (all Z: (all P: (all P1: (((Z = X - 1), (P1 = P-1), (P1 >= 0)) andC ((at(X,Y) x free(Z,Y) x power(P)) linImp (at(Z,Y) x free(X,Y) x power(P1))))))))),
move_left act (all X: (all Y: (all Z: (all P: (all P1: (((Z = Y - 1), (P1 = P-1), (P1 >= 0)) andC ((at(X,Y) x free(X,Z) x power(P)) linImp (at(X,Z) x free(X,Y) x power(P1)))))))))
],
Area = [power(3.0), at(1.0,1.0), free(1.0,2.0), free(2.0,2.0), free(3.0,2.0)],
Goal = at(3.0,2.0) x erase,
prove(Rules, Area, Goal, Actions).

Proof terms are:

Program starts..
(let_c (pair_c 3.0=2.0+1, 0.0=1.0-1, 0.0>=0, v15332) = (app_all((app_all((app_all((app_all((app_all(v15326, 2.0)), 2.0)), 3.0)), 1.0)), 0.0)) in (let_c (pair_c 2.0=1.0+1, 1.0=2.0-1, 1.0>=0, v16617) = (app_all((app_all((app_all((app_all((app_all(v16611, 1.0)), 2.0)), 2.0)), 2.0)), 1.0)) in (let_c (pair_c 2.0=1.0+1, 2.0=3.0-1, 2.0>=0, v17013) = (app_all((app_all((app_all((app_all((app_all(v17007, 1.0)), 1.0)), 2.0)), 3.0)), 2.0)) in (let_x (v17110 x v17111) = (appˆ(v17013, (v2 x (v3 x v1)))) in (let_x (v17112 x v17113) = v17111 in (let_x (v17138 x v17139) = (appˆ(v16617, (v17110 x (v4 x v17113))))) in (let_x (v17140 x v17141) = v17139 in (let_x (v17148 x v17149) = (appˆ(v15332, (v17138 x (v5 x v17141)))) in (let_x (v17150 x v17151) = v17149 in (v17148 x < >))))))))))

Information about time, entered predicates and actions are:

Total time is: 11.94
Counter is: 192580
Rules = [move_down act (all X:all Y:all Z:all... : ...), move_right act (all X:all Y:all Z:all...), move_up act (all X:all Y:all... : ...), move_left act (all X:all Y:all...)],
Area = [power(3.0), at(1.0, 1.0), free(1.0, 2.0), free(2.0, 2.0), free(3.0, 2.0)],
Goal = at(3.0, 2.0)x erase,
Actions = [move_right, move_down, move_down]

5.4.4 Mail Delivery Robot

Next problem domain is about the mail delivery robot. In the problem domain, there is a mobile robot collecting mails from office rooms and moving them into a common mail store. We must note that, the weight of mails vary. For the sake of less processing time, planning area is limited to three rooms. In three rooms, mails have $10kg$, $7kg$ and $3kg$ weights, respectively. Figure 5.3 illustrates the initial case of the planning environment.

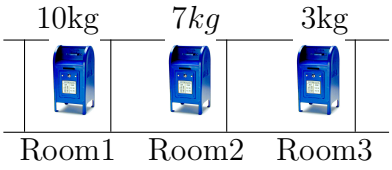

Figure 5.3: Mail delivery planning environment

In this planning domain, propositions are *weight*, *mail* and *power*. The proposition *weight(w)* denotes the total weight of mails which the robot is carrying. In the proposition *mail(x,w)*, x denotes the room number and w denotes the weight of the mail. Same as in the example 5.4.3, *power(p)* proposition denotes the total power of the robot. Formulation of the power consuming while picking up a mail is:

$$Final\ Power = Initial\ Power - (Initial\ Weight + Mail\ Weight)\ ,$$

where *Initial Weight* is the total weight of mails which the robot is carrying and *Mail Weight* is the weight of the mail, picking up by the robot. Emptying all mails on the robot decreases the power 5. On the following, we encode resources and the goal for SWI-Prolog code. Two unrestricted rules (actions) are:

$\Gamma = [$
pick mail : $\forall x.\forall w.\forall w_1.\forall k.\forall p_1.\forall p.(((k > 0), (w1 = w + k), (p_1 = p - w_1), (p_1 >= 0)) \wedge_c ((weight(w) \otimes mail(x, k) \otimes power(p)) \multimap (weight(w_1) \otimes mail(x, 0.0) \otimes power(p_1))))$,
empty mails : $\forall w.\forall p_1.\forall p.(((w > 0), (p_1 = p - 5), (p_1 >= 0)) \wedge_c ((weight(w) \otimes power(p)) \multimap (weight(0.0) \otimes power(p_1))))$
].

Initial resources for the example are:

$$\Delta_0 = [power(33.0), weight(0.0), mail(1.0, 10.0), mail(2.0, 7.0), mail(3.0, 3.0)].$$

The goal for this example is:

$$G = weight(0.0) \otimes mail(1.0, 0.0) \otimes mail(2.0, 0.0) \otimes mail(3.0, 0.0) \otimes T \ ,$$

meaning that the robot collects all mails from office rooms and empties them into the mail store. Running this example on Swi-Prolog with a depth limit 50, *CFRM* system constructs a plan with actions [pick mail, empty mails, pick mail, pick mail, empty mails]. With this resources and the goal, the least power requirement is 33. If we run the program with a power less than 33, the planner can not construct any plan. The plan of the planner is listed as:

1. The robot picks up the mail in Room1.
   $(Weight_{mail} = 10kg, Weight_{total} = 10kg, Power_{final} = 33 - 10 = 23)$
2. The robot empties the mail.
   $(Weight_{mail} = 0kg, Weight_{total} = 0kg, Power_{final} = 23 - 5 = 18)$
3. The robot picks up the mail in Room3.
   $(Weight_{mail} = 3kg, Weight_{total} = 3kg, Power_{final} = 18 - 3 = 15)$
4. The robot picks up the mail in Room2.
   $(Weight_{mail} = 7kg, Weight_{total} = 10kg, Power_{final} = 15 - 10 = 5)$
5. The robot empties the mail.
   $(Weight_{mail} = 0kg, Weight_{total} = 0kg, Power_{final} = 5 - 5 = 0)$

In this example, a proof is constructed in about 30 minutes and 1,852,338 predicates are entered during the proof search. When the problem domain gets bigger, the time for finding a proof takes longer. Thus, we show examples in small domains. However, when we increase the speed of the system, we can try different and more complex problems in huge domains.

Chapter 6

Conclusion And Future Work

Robotic planning and automation in continues domains are more challenging than discrete domains. Thus, many researchers work on planning for continues domains and uncertain environments. We are interested in logic based languages for robotic planning. However, one major problem for logic based systems is that the computational complexity of reasoning systems based on theorem proving increases with their expressivity. We believe that one of the best logical formalisms that can address robotic planning issues is *Intuitionistic First-Order Linear Logic*.

In this thesis, based on *FocLL* system, first we introduced a new automated theorem prover system, which we called *Full Resource Management system (FRM)*, for robotic planning in discrete domains. *FRM* system is an efficient backward sequent calculus for intuitionistic linear logic with focusing and resource management. Afterwards, for robotic planning in both discrete and continues domains, we incorporated constraints into *FRM* system, which we called this new system as *CFRM*. Using constraint solvers of SWI-Prolog helped us reducing the complexity associated with the encoding. However, some restrictions with constraints are still existing such as *case splitting* and *support for interpreted symbols during unification*. We handled the latter restriction with some encoding modifications of constraints. Since we wanted to record the constructed proof to yield corresponding plan to be used in our domain, we enriched *CFRM* system with proof terms that carry enough information to reconstruct deduction of the proof.

We tested our *CFRM* system for some robotic planning examples, where we introduced these planning examples in the thesis. One of the planning domain is about path finding among mines and the other one is mail delivery system. Our theorem prover system successfully achieved all goals. However, as we mentioned in the first chapter, when the example domain increases linearly, computational time increases exponentially. Hence, in the future, we are planning to either modify our SWI-Prolog implementation or restrict our language into *LHHF* for finding plans in a shorter time.

We also mentioned that *CFRM* system had some restrictions such as *case splitting* and *support for interpreted symbols during unification*. We can handle the latter restriction with some encoding modifications of constraints. However, we also need to handle *case splitting* for the soundness and completeness of *CFRM* system. To this end, we are planning to modify our proof system for proving the soundness and completeness of the system. We expect that this will help us a better understanding of the *CFRM* system behavior.

As we mentioned in 5.4.3, we can not define the negation of any resource in the planning domain. However, as a future work, we are planning to incorporate encoding the negation of any resource into CFRM system. In [4], Chaudhuri introduced *possibility* concept for encoding the negation of linear resources. To this end, we can use the same approach as in the dissertation of Chaudhuri.

Our next step will be applying experiences gained from this thesis into a real mobile robot. We suppose that working on planning for real robots will be more challenging than working on simulations, however, gained experiences will be priceless. Our ultimate goal is building a reliable and fully automated robotic planning system.

Bibliography

[1] http://marsrovers.nasa.gov/home/index.html.

[2] I. Bratko. *Prolog programming for artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[3] I. Cervesato, J. S. Hodas, and F. Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1-2):133–163, Feb. 2000.

[4] K. Chaudhuri. *The Focused Inverse Method for Linear Logic*. Phd, Carnegie Mellon University, Pittsburgh, PA, December 2006.

[5] D. C. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G. J. Pappas. Valet parking without a valet. pages 572–577, October 2007.

[6] D. C. Dennett. *Brainstorms: Philosophical essays on mind and psychology*. Cambridge, Mass.: Bradford Books/MIT Press, 1978.

[7] M. B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132:151–182, 2001.

[8] T. Douillard and C. Jermann. Splitting heuristics for disjunctive numerical constraints. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 140–144, New York, NY, USA, 2008. ACM.

[9] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[10] M. Fitting. *First-order logic and automated theorem proving.* Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[11] G. Gentzen. *Untersuchungen uber das logische SchlieBen.* PhD thesis, University of Gottingen, 1935.

[12] H. K.-G. Georgios E. Fainekos and G. J. Pappas. Temporal logic motion planning for mobile robots. 2005.

[13] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[14] H. Goguen, S. Combinators, and J. Goubault-Larrecq. Sequent combinators: A hilbert system for the lambda calculus, 1999.

[15] B. Goldstein and R. Shotwell. Phoenix: The first mars scout mission. In *Aerospace conference, 2009 IEEE*, pages 1–20, March 2009.

[16] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artif. Intell.*, 56(2-3):223–254, 1992.

[17] J. Herbrand. *Logical Writings.* Harvard University Press, 1971. Edited by Warren D. Goldfarb.

[18] J. S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation.* PhD thesis, University of Pennsylvania, 1994.

[19] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110:32–42, 1994.

[20] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, New York, NY, USA, 2004.

[21] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.

[22] J. Jaffar and S. Michaylov. Methodology and implementation of a clp system. In J.-L. Lassez, editor, *Logic Programming: Proc. of the Fourth International Conference (Volume 1)*, pages 196–218. MIT Press, Cambridge, MA, 1987.

[23] H. J. Levesque, R. Reiter, Y. Lesprance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31, 1997.

[24] J. marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2:297–347, 1992.

[25] M. C. Mayer, C. Limongelli, A. Orlandini, and V. Poggioni. Linear temporal logic as an executable semantics for planning languages. *J. of Logic, Lang. and Inf.*, 16(1):63–89, 2007.

[26] J. McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, pages 33–70. North-Holland, 1963.

[27] S. Michaylov. *Design and implementation of practical constraint logic programming systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[28] V. Nr, H. Levesque, C. F. H. Levesque, F. Pirri, F. Pirri, R. Reiter, and R. Reiter. Foundations for the situation calculus, 1998.

[29] R. A. O'Keefe. *The craft of Prolog*. MIT Press, Cambridge, MA, USA, 1990.

[30] P. P. Parra, E. Yescas, and J. Vásquez. Planning using situation calculus, prolog and a mobile robot. In *LA-NMR*, 2007.

[31] F. Pfenning. Lecture notes on linear logic. Technical report, Carnegie Mellon University, 2002.

[32] F. Pfenning. Lecture notes on automated theorem proving. Technical report, Carnegie Mellon University, 2004.

[33] M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *In Proc. IJCAI'01*, pages 479–484. AAAI Press, 2001.

[34] S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Logic*, 7(4):723–748, 2006.

[35] R. Reiter. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. MIT Press, Cambridge, Mass., 2001. The frame problem and the situation calculus.

[36] J. A. Robinson. Computational logic: The unification computation. *Machine Intelligence*, 6:63–72, 1971.

[37] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[38] R. M. Smullyan. *First-Order Logic*. Dover Publications, Inc., 1995.

[39] M. Thielscher. From situation calculus to fluent calculus: state update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299, 1999.

[40] M. Thielscher. Flux: A logic programming method for reasoning agents. *Theory Pract. Log. Program.*, 5(4-5):533–565, 2005.

[41] F. P. Uluc Saranli. Using constrained intuitionistic linear logic for hybrid robotic planning problems. 2007.

Appendix A

Sequent Calculus for Linear Logic

$$\frac{}{\Gamma; A \Longrightarrow A}\ \mathit{init} \qquad \frac{(\Gamma, A); (\Delta, A) \Longrightarrow C}{(\Gamma, A); \Delta \Longrightarrow C}\ \mathit{copy}$$

Figure A.1: Hypotheses

$$\frac{\Gamma; \Delta, A \Longrightarrow B}{\Gamma; \Delta \Longrightarrow A \multimap B} \multimap R \qquad \frac{\Gamma; \Delta_1 \Longrightarrow A \qquad \Gamma; \Delta_2, B \Longrightarrow C}{\Gamma; \Delta_1, \Delta_2, A \multimap B \Longrightarrow C} \multimap L$$

$$\frac{\Gamma; \Delta_1 \Longrightarrow A \qquad \Gamma; \Delta_2 \Longrightarrow B}{\Gamma; \Delta_1, \Delta_2 \Longrightarrow A \otimes B} \otimes R \qquad \frac{\Gamma; \Delta, A, B \Longrightarrow C}{\Gamma; \Delta, A \otimes B \Longrightarrow C} \otimes L$$

$$\frac{}{\Gamma; \cdot \Longrightarrow 1}\ 1R \qquad \frac{\Gamma; \Delta \Longrightarrow C}{\Gamma; \Delta, 1 \Longrightarrow C}\ 1L$$

Figure A.2: Multiplicative Connectives

$$\frac{\Gamma;\Delta \Longrightarrow A \qquad \Gamma;\Delta \Longrightarrow B}{\Gamma;\Delta \Longrightarrow A\&B}\ \&R$$

$$\frac{\Gamma;\Delta, A \Longrightarrow C}{\Gamma;\Delta, A\&B \Longrightarrow C}\ \&L_1 \qquad \frac{\Gamma;\Delta, B \Longrightarrow C}{\Gamma;\Delta, A\&B \Longrightarrow C}\ \&L_2$$

$$\frac{}{\Gamma;\Delta \Longrightarrow T}\ TR \qquad \text{No T left rule}$$

$$\frac{\Gamma;\Delta \Longrightarrow A}{\Gamma;\Delta \Longrightarrow A \oplus B}\ \oplus R_1 \qquad \frac{\Gamma;\Delta \Longrightarrow B}{\Gamma;\Delta \Longrightarrow A \oplus B}\ \oplus R_2$$

$$\frac{\Gamma;\Delta, A \Longrightarrow C \qquad \Gamma;\Delta, B \Longrightarrow C}{\Gamma;\Delta, A \oplus B \Longrightarrow C}\ \oplus L$$

$$\text{No 0 right rule} \qquad \frac{}{\Gamma;\Delta, 0 \Longrightarrow C}\ 0L$$

Figure A.3: Additive Connectives

$$\frac{\Gamma;\Delta \Longrightarrow [a/x]A}{\Gamma;\Delta \Longrightarrow \forall x.A}\ \forall R^a \qquad \frac{\Gamma;\Delta, [t/x]A \Longrightarrow C}{\Gamma;\Delta, \forall x.A \Longrightarrow C}\ \forall L$$

$$\frac{\Gamma;\Delta \Longrightarrow [t/x]A}{\Gamma;\Delta \Longrightarrow \exists x.A}\ \exists R \qquad \frac{\Gamma;\Delta, [a/x]A \Longrightarrow C}{\Gamma;\Delta, \exists x.A \Longrightarrow C}\ \exists L^a$$

Figure A.4: Quantifiers

$$\frac{(\Gamma, A);\Delta \Longrightarrow B}{\Gamma;\Delta \Longrightarrow A \supset B}\ \supset R \qquad \frac{\Gamma;\cdot \Longrightarrow A \qquad \Gamma;\Delta, B \Longrightarrow C}{\Gamma;\Delta, A \supset B \Longrightarrow C}\ \supset L$$

$$\frac{\Gamma;\cdot \Longrightarrow A}{\Gamma;\cdot \Longrightarrow !A}\ !R \qquad \frac{(\Gamma, A);\Delta \Longrightarrow C}{\Gamma;(\Delta, !A) \Longrightarrow C}\ !L$$

Figure A.5: Exponentials

Appendix B

Focused Intuitionistic First-Order Linear Logic (FocLL)

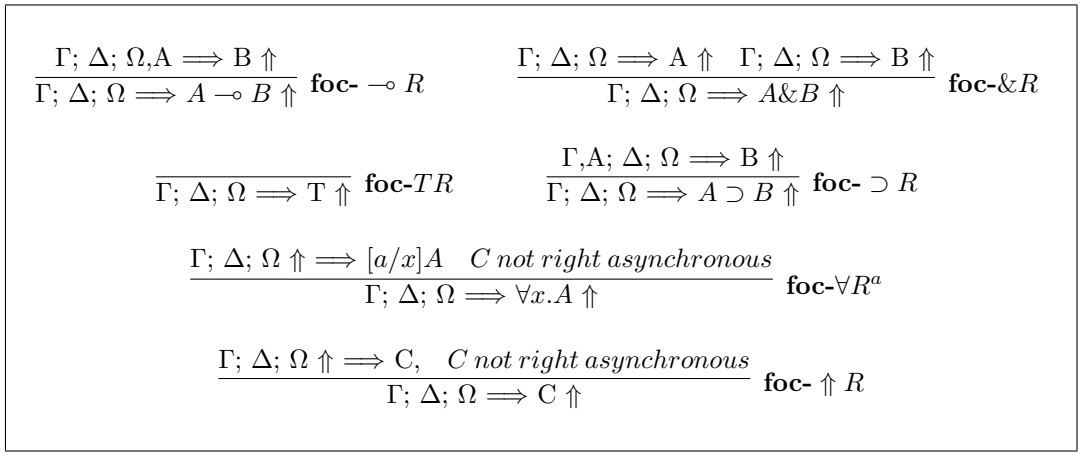

Figure B.1: Right invertible rules for the FocLL system

$$\frac{\Gamma;\ \Delta;\ \Omega,\mathrm{A},\mathrm{B} \Uparrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \Omega,\mathrm{A}\otimes\mathrm{B} \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\otimes L \qquad \frac{\Gamma;\ \Delta;\ \Omega \Uparrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \Omega,1 \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}1L$$

$$\frac{\Gamma;\ \Delta;\ \Omega,\mathrm{A} \Uparrow \Longrightarrow \mathrm{C} \quad \Gamma;\ \Delta;\ \Omega,\mathrm{B} \Uparrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \Omega,\mathrm{A}\oplus\mathrm{B} \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\oplus L \qquad \frac{}{\Gamma;\ \Delta;\ \Omega,0 \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}0L$$

$$\frac{\Gamma,\mathrm{A};\ \Delta;\ \Omega \Uparrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \Omega,!\mathrm{A} \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}!L \qquad \frac{\Gamma;\ \Delta;\ \Omega,[\mathrm{a}/\mathrm{x}]\mathrm{A} \Uparrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \Omega,\exists\mathrm{x}.\mathrm{A} \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\exists L^{a}$$

$$\frac{\Gamma;\ \Delta,\mathrm{A};\ \Omega \Uparrow \Longrightarrow \mathrm{C}, \quad A\ not\ left\ asynchronous}{\Gamma;\ \Delta;\ \Omega,\mathrm{A} \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-} \Uparrow L$$

Figure B.2: Left invertible rule set for FocLL system

$$\frac{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{C} \Downarrow, \quad C\ not\ atomic}{\Gamma;\ \Delta;\ \cdot \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}decideR$$

$$\frac{\Gamma;\ \Delta;\ \mathrm{A} \Downarrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta,\mathrm{A};\ \cdot \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}decideL \qquad \frac{\Gamma,\mathrm{A};\ \Delta;\ \mathrm{A} \Downarrow \Longrightarrow \mathrm{C}}{\Gamma,\mathrm{A};\ \Delta;\ \cdot \Uparrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}decideL!$$

Figure B.3: Decision rule set for FocLL system

$$\frac{\Gamma;\ \Delta_1;\ \cdot \Longrightarrow \mathrm{A} \Downarrow \quad \Gamma;\ \Delta_2;\ \cdot \Longrightarrow \mathrm{B} \Downarrow}{\Gamma;\ \Delta_1,\Delta_2;\ \cdot \Longrightarrow \mathrm{A}\otimes\mathrm{B} \Downarrow}\ \textbf{foc-}\otimes R \qquad \frac{}{\Gamma;\ \cdot;\ \cdot \Longrightarrow 1 \Downarrow}\ \textbf{foc-}1R$$

$$\frac{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{A} \Downarrow}{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{A}\oplus\mathrm{B} \Downarrow}\ \textbf{foc-}\oplus R_1 \qquad \frac{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{B} \Downarrow}{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{A}\oplus\mathrm{B} \Downarrow}\ \textbf{foc-}\oplus R_2$$

$$\frac{\Gamma;\ \Delta;\ \cdot \Longrightarrow [\mathrm{t}/\mathrm{x}]\mathrm{A} \Downarrow}{\Gamma;\ \Delta;\ \cdot \Longrightarrow \exists\mathrm{x}.\mathrm{A} \Downarrow}\ \textbf{foc-}\exists R^{a} \qquad \frac{\Gamma;\ \cdot;\ \cdot \Longrightarrow \mathrm{A} \Uparrow}{\Gamma;\ \cdot;\ \cdot \Longrightarrow !\mathrm{A} \Downarrow}\ \textbf{foc-}!R$$

Figure B.4: Right focusing rule set for FocLL system

$$\frac{\Gamma;\ \Delta_2;\ \mathrm{B} \Downarrow \Longrightarrow \mathrm{C} \quad \Gamma;\ \Delta_1;\ \cdot \Longrightarrow \mathrm{A} \Downarrow}{\Gamma;\ \Delta_1, \Delta_2;\ \mathrm{A} \multimap \mathrm{B} \Downarrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\ \multimap L$$

$$\frac{\Gamma;\ \Delta;\ \mathrm{B} \Downarrow \Longrightarrow \mathrm{C} \quad \Gamma;\ \cdot;\ \cdot \Longrightarrow \mathrm{A} \Uparrow}{\Gamma;\ \Delta;\ A \supset B \Downarrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\ \supset L$$

$$\frac{\Gamma;\ \Delta;\ \mathrm{A} \Downarrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \mathrm{A\&B} \Downarrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\&L_1 \qquad \frac{\Gamma;\ \Delta;\ \mathrm{B} \Downarrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \mathrm{A\&B} \Downarrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\&L_2$$

$$\text{no left rule for T} \qquad \frac{\Gamma;\ \Delta;\ [\mathrm{t/x}]\mathrm{A} \Downarrow \Longrightarrow \mathrm{C}}{\Gamma;\ \Delta;\ \forall \mathrm{x.A} \Downarrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\forall L$$

$$\frac{}{\Gamma;\ \cdot;\ \mathrm{P} \Downarrow \Longrightarrow \mathrm{P}}\ \textbf{foc-}\mathit{init}$$

$$\frac{\Gamma;\ \Delta;\ \mathrm{A} \Uparrow \Longrightarrow \mathrm{C} \quad \textit{A not atomic and not left synchronous}}{\Gamma;\ \Delta;\ \mathrm{A} \Downarrow \Longrightarrow \mathrm{C}}\ \textbf{foc-}\Downarrow L$$

$$\frac{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{A} \Uparrow}{\Gamma;\ \Delta;\ \cdot \Longrightarrow \mathrm{A} \Downarrow}\ \textbf{foc-}\Downarrow R$$

Figure B.5: Left focusing rule set for FocLL system

Appendix C

Soundness and Completeness of The FRM System

C.1 Key Properties of The FRM System

Certain properties of the FRM system will be useful in proving its soundness and completeness with respect to the FocLL system.

Lemma C.1.1 *(Subcontext property for **FRM**).*

- *If* Γ*;* $\Delta_I \setminus \Delta_O$*;* $\Omega \Longrightarrow_v \ G \Uparrow$*, then* $\Delta_O \subseteq \Delta_I$ *.*
- *If* Γ*;* $\Delta_I \setminus \Delta_O$*;* $\Omega \Uparrow \Longrightarrow_v \ G$*, then* $\Delta_O \subseteq \Delta_I$ *.*
- *If* Γ*;* $\Delta_I \setminus \Delta_O$*;* $\cdot \Longrightarrow_v \ G \Downarrow$*, then* $\Delta_O \subseteq \Delta_I$ *.*
- *If* Γ*;* $\Delta_I \setminus \Delta_O$*;* $A \Downarrow \Longrightarrow_v \ G$*, then* $\Delta_O \subseteq \Delta_I$ *.*

Proof. The proof proceeds by nested structural induction on the following derivations

$$\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ \mathrm{G} \Uparrow ,$$
$$\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \ \mathrm{G} ,$$
$$\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_v\ \ \mathrm{G} \Downarrow ,$$
$$\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ \ \mathrm{G} .$$

We show some key cases that are nontrivial to prove. Other cases that are not shown here are either trivial or similar to the ones proven below.

Case $\&\mathbf{R}_{01}$: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{cc}\mathfrak{F}_1 & \mathfrak{F}_2\\ \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A \Uparrow & \Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega \Longrightarrow_1\ \ B \Uparrow\end{array}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A\&B \Uparrow}\ \&R_{01}$$

$\Delta_O \subseteq \Delta_I$ By i.h. on $\mathfrak{F}_1$

Case $\&\mathbf{R}_{11}$: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{cc}\mathfrak{F}_1 & \mathfrak{F}_2\\ \Gamma;\ \Delta_I \setminus \Delta_{O1};\ \Omega \Longrightarrow_1\ \ A \Uparrow & \Gamma;\ \Delta_I \setminus \Delta_{O2};\ \Omega \Longrightarrow_1\ \ B \Uparrow\end{array}}{\Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega \Longrightarrow_1\ \ A\&B \Uparrow}\ \&R_{11}$$

$\Delta_{O1} \subseteq \Delta_I$ By i.h. on $\mathfrak{F}_1$

$\Delta_{O2} \subseteq \Delta_I$ By i.h. on $\mathfrak{F}_2$

$\Delta_{O1} \cap \Delta_{O2} \subseteq \Delta_I$ By multiset properties

Case $\oplus\mathbf{L}_{01}$: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{cc}\mathfrak{F}_1 & \mathfrak{F}_2\\ \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ \ C & \Gamma;\ \Delta_I \setminus \Delta_{O2};\ \Omega, B \Uparrow \Longrightarrow_1\ \ C\end{array}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ \ C}\ \oplus L_{01}$$

$\Delta_O \subseteq \Delta_I$ By i.h. on $\mathfrak{F}_1$

Case $\Uparrow \mathbf{L}_{1A}$: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{c}\mathfrak{F}_1 \\ \Gamma;\ \Delta_I, A \setminus \Delta_O, A;\ \Omega \Uparrow \Longrightarrow_1\ C,\ A\ not\ left\ asynchronous\end{array}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_1\ C} \Uparrow L_{1A}$$

$\Delta_O, A \subseteq \Delta_I, A$ By i.h. on $\mathfrak{F}_1$

$\Delta_O \subseteq \Delta_I$ By multiset properties, deleting A from both sides

Case $\Uparrow \mathbf{L}_v$: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{c}\mathfrak{F}_1 \\ \Gamma;\ \Delta_I, A \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ C,\quad A\ not\ left\ asynchronous\end{array}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_v\ C} \Uparrow L_v$$

$A \notin \Delta_O$ We know that A is not in the output. L_{1A} holds counter case

$\Delta_O \subseteq \Delta_I, A$ By i.h. on $\mathfrak{F}_1$

$\Delta_O \subseteq \Delta_I$ By multiset properties, if we delete A, it still holds

Case decide $-$ **L**: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{c}\mathfrak{F}_1 \\ \Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_v\ C\end{array}}{\Gamma;\ \Delta_I, A \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ C}\ decide - L$$

$\Delta_O \subseteq \Delta_I$ By i.h. on $\mathfrak{F}_1$

$\Delta_O \subseteq \Delta_I, A$ By multiset properties

Case $\otimes$**R**: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{cc}\mathfrak{F}_1 & \mathfrak{F}_2 \\ \Gamma;\, \Delta_I \setminus \Delta_M;\, \cdot \Longrightarrow_u\ A \Downarrow & \Gamma;\, \Delta_M \setminus \Delta_O;\, \cdot \Longrightarrow_w\ B \Downarrow\end{array}}{\Gamma;\, \Delta_I \setminus \Delta_O;\, \cdot \Longrightarrow_{u \vee w}\ A \otimes B \Downarrow}\ \otimes R$$

$\Delta_M \subseteq \Delta_I$	By i.h. on $\mathfrak{F}_1$
$\Delta_O \subseteq \Delta_M$	By i.h. on $\mathfrak{F}_2$
$\Delta_O \subseteq \Delta_I$	Since $\Delta_O \subseteq \Delta_M \subseteq \Delta_I$

Case $\multimap$ L: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{cc}\mathfrak{F}_1 & \mathfrak{F}_2 \\ \Gamma;\, \Delta_I \setminus \Delta_M;\, B \Downarrow \Longrightarrow_u\ C & \Gamma;\, \Delta_M \setminus \Delta_O;\, \cdot \Longrightarrow_w\ A \Downarrow\end{array}}{\Gamma;\, \Delta_I \setminus \Delta_O;\, A \multimap B \Downarrow \Longrightarrow_{u \vee w}\ C}\ \multimap L$$

$\Delta_M \subseteq \Delta_I$	By i.h. on $\mathfrak{F}_1$
$\Delta_O \subseteq \Delta_M$	By i.h. on $\mathfrak{F}_2$
$\Delta_O \subseteq \Delta_I$	Since $\Delta_O \subseteq \Delta_M \subseteq \Delta_I$

□

C.2 Soundness

Theorem C.2.1 *(Soundness of **FRM** with respect to **FocLL**).*

- *If* $\Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_0\ G \Uparrow$, *then* $\Gamma;\, \Delta_I - \Delta_O;\, \Omega \Longrightarrow G \Uparrow$.
- *If* $\Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Longrightarrow_1\ G \Uparrow$, *then* $\Gamma;\, (\Delta_I - \Delta_O, \Delta');\, \Omega \Longrightarrow G \Uparrow$ *for every context* $\Delta' \subseteq \Delta_O$.
- *If* $\Gamma;\, \Delta_I \setminus \Delta_O;\, \Omega \Uparrow \Longrightarrow_0\ G$, *then* $\Gamma;\, \Delta_I - \Delta_O;\, \Omega \Uparrow \Longrightarrow G$.

- *If* $\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_1\ \ G$, *then* $\Gamma;\ (\Delta_I - \Delta_O, \Delta');\ \Omega \Uparrow \Longrightarrow G$ *for every context* $\Delta' \subseteq \Delta_O$.
- *If* $\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_0\ \ G \Downarrow$, *then* $\Gamma;\ \Delta_I - \Delta_O;\ \cdot \Longrightarrow G \Downarrow$.
- *If* $\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_1\ \ G \Downarrow$, *then* $\Gamma;\ (\Delta_I - \Delta_O, \Delta');\ \cdot \Longrightarrow G \Downarrow$ *for every context* $\Delta' \subseteq \Delta_O$.
- *If* $\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_0\ \ G$, *then* $\Gamma;\ \Delta_I - \Delta_O;\ A \Downarrow \Longrightarrow G$.
- *If* $\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_1\ \ G$, *then* $\Gamma;\ (\Delta_I - \Delta_O, \Delta');\ A \Downarrow \Longrightarrow G$ *for every context* $\Delta' \subseteq \Delta_O$.

Proof. The proof proceeds by nested structural induction on the following derivations

$\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ \mathrm{G} \Uparrow$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_1\ \ \mathrm{G} \Uparrow$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_0\ \ \mathrm{G}$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_1\ \ \mathrm{G}$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_0\ \ \mathrm{G} \Downarrow$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ \cdot \Longrightarrow_1\ \ \mathrm{G} \Downarrow$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_0\ \ \mathrm{G}$,

$\Gamma;\ \Delta_I \setminus \Delta_O;\ A \Downarrow \Longrightarrow_1\ \ \mathrm{G}$.

We show some key cases of the proof.

Case $\multimap \mathbf{R}_v$: Suppose that the given derivation ends with the rule

$$\frac{\begin{array}{c}\mathfrak{F}_1 \\ \Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Longrightarrow_v\ \ B \Uparrow\end{array}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ A \multimap B \Uparrow} \multimap R_v$$

For v = 0

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega, A \Longrightarrow B \Uparrow$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Longrightarrow A \multimap B \Uparrow$ By rule **foc-**$\multimap R$

For v = 1

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega, A \Longrightarrow B \Uparrow$ By i.h. on $\mathfrak{F}_1$, $\forall.\Delta' \subseteq \Delta_O$

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Longrightarrow A \multimap B \Uparrow$ By rule **foc-**$\multimap R$

Case $\&\mathbf{R}_{01}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ A \Uparrow} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega \Longrightarrow_1\ B \Uparrow}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_0\ A\&B \Uparrow}\ \&R_{01}$$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Longrightarrow A \Uparrow$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_I - (\Delta_O, \Delta_2), \Delta';\ \Omega \Longrightarrow B \Uparrow$ By i.h. on $\mathfrak{F}_2$, $\forall.\Delta' \subseteq (\Delta_O, \Delta_2)$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Longrightarrow A\&B \Uparrow$ By rule **foc-**$\&R$ for $\Delta' = \Delta_2$

Case $\&\mathbf{R}_{10}$ is symmetric to $\&\mathbf{R}_{01}$.

Case $\&\mathbf{R}_{11}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta_I \setminus \Delta_{O1};\ \Omega \Longrightarrow_1\ A \Uparrow} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_I \setminus \Delta_{O2};\ \Omega \Longrightarrow_1\ B \Uparrow}}{\Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega \Longrightarrow_1\ A\&B \Uparrow}\ \&R_{11}$$

$\Gamma;\ \Delta_I - \Delta_{O1}, \Delta_1';\ \Omega \Longrightarrow A \Uparrow$ By i.h. on $\mathfrak{F}_1$ $\forall\ .\Delta_1' \subseteq \Delta_{O1}$

$\Gamma;\ \Delta_I - \Delta_{O2}, \Delta_2';\ \Omega \Longrightarrow B \Uparrow$ By i.h. on $\mathfrak{F}_2$ $\forall\ .\Delta_2' \subseteq \Delta_{O2}$

$\Gamma;\ \Delta_I - (\Delta_{O1} \cap \Delta_{O2}), \Delta_3';\ \Omega \Longrightarrow A\&B \Uparrow$ By rule **foc-**$\&R$, $\forall.\Delta_3' \subseteq (\Delta_{O1} \cap \Delta_{O2})$

Case $T\mathbf{R}$: Suppose that the given derivation ends with the rule

$$\frac{}{\Gamma;\ \Delta_I \setminus \Delta_I;\ \Omega \Longrightarrow_1\ \ T \Uparrow}\ TR$$

$\Gamma;\ \Delta';\ \Omega \Longrightarrow T \Uparrow$ By rule **foc-**TR, $\forall.\Delta' \subseteq \Delta_I$

Case $\Uparrow$ **R**: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \ C},\ C \textit{ not right asynchronous}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ \ C \Uparrow}\ \Uparrow R$$

For v = 0

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Longrightarrow C \Uparrow$ By rule **foc-**$\Uparrow R$

For v = 1

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta' \subseteq \Delta_O$

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Longrightarrow C \Uparrow$ By By rule **foc-**$\Uparrow R$, $\forall.\Delta' \subseteq \Delta_O$

Case $\oplus\mathbf{L}_{01}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ \ C} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_I \setminus \Delta_O, \Delta_2;\ \Omega, B \Uparrow \Longrightarrow_1\ \ C}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \oplus B \Uparrow \Longrightarrow_0\ \ C}\ \&R_{01}$$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega, A \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_I - (\Delta_O, \Delta_2), \Delta';\ \Omega, B \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_2\ \forall\ .\Delta' \subseteq \Delta_O, \Delta_2$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Uparrow \Longrightarrow A \oplus B$ By rule **foc-**$\oplus R$ for $\Delta' = \Delta_2$

Case $\oplus\mathbf{L}_{11}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta_I \setminus \Delta_{O1};\ \Omega, A \Longrightarrow_1\ \ C \Uparrow} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_I \setminus \Delta_{O2};\ \Omega, B \Longrightarrow_1\ \ C \Uparrow}}{\Gamma;\ \Delta_I \setminus \Delta_{O1} \cap \Delta_{O2};\ \Omega, A \oplus B \Longrightarrow_1\ \ C \Uparrow}\ \&L_{11}$$

$\Gamma;\ \Delta_I - \Delta_{O1}, \Delta_1';\ \Omega \Longrightarrow A \Uparrow$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta_1' \subseteq \Delta_{O1}$

$\Gamma;\ \Delta_I - \Delta_{O2}, \Delta_2';\ \Omega \Longrightarrow B \Uparrow$ By i.h. on $\mathfrak{F}_2\ \forall\ .\Delta_2' \subseteq \Delta_{O2}$

$\Gamma;\ \Delta_I - (\Delta_{O1} \cap \Delta_{O2}), \Delta_3';\ \Omega \Longrightarrow A\&B \Uparrow$ By rule **foc-**$\&R$, $\forall.\Delta_3' \subseteq (\Delta_{O1} \cap \Delta_{O2})$

Case $\Uparrow$ **R**: Suppose that the given derivation ends with the rule

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_v\ \overset{\mathfrak{F}_1}{C},\ C \textit{ not right asynchronous}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega \Longrightarrow_v\ C \Uparrow} \Uparrow R$$

For v = 0

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_I - \Delta_O;\ \Omega \Longrightarrow C \Uparrow$ By By rule **foc-**$\Uparrow R$

For v = 1

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta' \subseteq \Delta_O$

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Longrightarrow C \Uparrow$ By By rule **foc-**$\Uparrow R$, $\forall.\Delta' \subseteq \Delta_O$

Case $\Uparrow$ **L** $_{1A}$: Suppose that the given derivation ends with the rule

$$\frac{\Gamma;\ \Delta_I, A \setminus \Delta_O, A;\ \Omega \Uparrow \Longrightarrow_1\ \overset{\mathfrak{F}_1}{C},\ A \textit{ not left asynchronous}}{\Gamma;\ \Delta_I \setminus \Delta_O;\ \Omega, A \Longrightarrow_1\ C \Uparrow} \Uparrow L_{1A}$$

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Uparrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta' \subseteq (\Delta_O, A)$

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ \Omega \Longrightarrow C \Uparrow$ By rule **foc-**$\Uparrow L_{1A}$

Case decideL: Suppose that the given derivation ends with the rule

$$\frac{\Gamma;\ \Delta_I \setminus \Delta_O;\ \overset{\mathfrak{F}_1}{A} \Downarrow \Longrightarrow_v\ C}{\Gamma;\ \Delta_I, A \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_v\ C}\ decideL$$

For v = 0

$\Gamma;\ \Delta_I - \Delta_O;\ A \Downarrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_I - \Delta_O, A;\ \cdot \Uparrow \Longrightarrow C$ By rule **foc-***decideL*

For v = 1

$\Gamma;\ \Delta_I - \Delta_O, \Delta';\ A \Downarrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta' \subseteq \Delta_O$

$\Gamma;\ \Delta_I - \Delta_O, \Delta', A;\ \cdot \Uparrow \Longrightarrow C$ By rule **foc-***decideL*

Case $\otimes$**R**: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta, \Delta_M, \Delta_O \setminus \Delta_M, \Delta_O;\ \cdot \Longrightarrow_u\ A \Downarrow} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_M, \Delta_O \setminus \Delta_O;\ \cdot \Longrightarrow_w\ B \Downarrow}}{\Gamma;\ \Delta, \Delta_M, \Delta_O \setminus \Delta_O;\ \cdot \Longrightarrow_{u \vee w}\ A \otimes B \Downarrow} \otimes R$$

For vw = 10

$\Gamma;\ \Delta, \Delta';\ \cdot \Longrightarrow A \Downarrow$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta' \subseteq \Delta_M, \Delta_O$

$\Gamma;\ \Delta_M;\ \cdot \Longrightarrow B \Downarrow$ By i.h. on $\mathfrak{F}_2$

$\Gamma;\ \Delta, \Delta_M, \Delta';\ \cdot \Longrightarrow A \otimes B \Downarrow$ By rule **foc-**$\otimes R$, $\forall\ .\Delta' \subseteq \Delta_O$

For vw = 00

$\Gamma;\ \Delta;\ \cdot \Longrightarrow A \Downarrow$ By i.h. on $\mathfrak{F}_1$

$\Gamma;\ \Delta_M;\ \cdot \Longrightarrow B \Downarrow$ By i.h. on $\mathfrak{F}_2$

$\Gamma;\ \Delta, \Delta_M;\ \cdot \Longrightarrow A \otimes B \Downarrow$ By rule **foc-**$\otimes R$

Case $\multimap$ **L**: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta, \Delta_M, \Delta_O \setminus \Delta_M, \Delta_O;\ \cdot \Longrightarrow_u\ A \Downarrow} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_M, \Delta_O \setminus \Delta_O;\ B \Downarrow \Longrightarrow_w\ C}}{\Gamma;\ \Delta, \Delta_M, \Delta_O \setminus \Delta_O;\ A \multimap B \Downarrow \Longrightarrow_{u \vee w}\ C} \multimap L$$

For vw = 10

$\Gamma;\ \Delta, \Delta';\ \cdot \Longrightarrow A \Downarrow$ By i.h. on $\mathfrak{F}_1\ \forall\ .\Delta' \subseteq \Delta_M, \Delta_O$

$\Gamma;\ \Delta_M;\ B \Downarrow \Longrightarrow C$ By i.h. on $\mathfrak{F}_2$

$\Gamma;\ \Delta, \Delta_M, \Delta';\ A \multimap B \Downarrow \Longrightarrow C$ By rule **foc-**$\multimap L$, $\forall\ .\Delta' \subseteq \Delta_O$

□

C.3 Completeness

Theorem C.3.1 *(Completeness of **FRM** with respect to **FocLL**).*

- *If* $\Gamma;\ \Delta;\ \Omega \Longrightarrow G \Uparrow$, *then*

  either $\Gamma;\ (\Delta, \Delta_O) \setminus \Delta_O;\ \Omega \Longrightarrow_0 \ G \Uparrow$ *for every context* Δ_O

  or $\Gamma;\ (\Delta, \Delta_O) \setminus (\Delta', \Delta_O);\ \Omega \Longrightarrow_1 \ G \Uparrow$ *for every context* Δ_O *and for some* $\Delta' \subseteq \Delta$.

- *If* $\Gamma;\ \Delta;\ \Omega \Uparrow \Longrightarrow G$, *then*

  either $\Gamma;\ (\Delta, \Delta_O) \setminus \Delta_O;\ \Omega \Uparrow \Longrightarrow_0 \ G$ *for every context* Δ_O

  or $\Gamma;\ (\Delta, \Delta_O) \setminus (\Delta', \Delta_O);\ \Omega \Uparrow \Longrightarrow_1 \ G$ *for every context* Δ_O *and for some* $\Delta' \subseteq \Delta$.

- *If* $\Gamma;\ \Delta;\ \cdot \Longrightarrow G \Downarrow$, *then*

  either $\Gamma;\ (\Delta, \Delta_O) \setminus \Delta_O;\ \cdot \Longrightarrow_0 \ G \Downarrow$ *for every context* Δ_O

  or $\Gamma;\ (\Delta, \Delta_O) \setminus (\Delta', \Delta_O);\ \cdot \Longrightarrow_1 \ G \Downarrow$ *for every context* Δ_O *and for some* $\Delta' \subseteq \Delta$.

- *If* $\Gamma;\ \Delta;\ A \Downarrow \Longrightarrow G$, *then*

  either $\Gamma;\ (\Delta, \Delta_O) \setminus \Delta_O;\ A \Downarrow \Longrightarrow_0 \ G$ *for every context* Δ_O

  or $\Gamma;\ (\Delta, \Delta_O) \setminus (\Delta', \Delta_O);\ A \Downarrow \Longrightarrow_1 \ G$ *for every context* Δ_O *and for some* $\Delta' \subseteq \Delta$.

Proof. By induction on the structure of a derivation of

$\Gamma;\ \Delta;\ \Omega \Longrightarrow \mathrm{G} \Uparrow$,

$\Gamma; \Delta; \Omega \Uparrow \Longrightarrow \mathrm{G}$,

$\Gamma; \Delta; \cdot \Longrightarrow \mathrm{G} \Downarrow$,

$\Gamma; \Delta; A \Downarrow \Longrightarrow \mathrm{G}$. We show some cases of the proof.

Case $foc- \multimap \mathbf{R}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma; \Delta; \Omega, A \Longrightarrow B \Uparrow}}{\Gamma; \Delta; \Omega \Longrightarrow A \multimap B \Uparrow} \; foc- \multimap R$$

Either

$\Gamma; \Delta, \Delta_O \setminus \Delta_O; \Omega, A \Longrightarrow_0 \; B \Uparrow$ By i.h. on $\mathfrak{F}_1 \; \forall \; .\Delta_O$

$\Gamma; \Delta, \Delta_O \setminus \Delta_O; \Omega \Longrightarrow_0 \; A \multimap B \Uparrow$ By rule $\multimap R_v$

or

$\Gamma; \Delta, \Delta_O \setminus \Delta', \Delta_O; \Omega, A \Longrightarrow_1 \; B \Uparrow$ By i.h. on $\mathfrak{F}_1 \; \forall \; .\Delta_O$ and for some $\Delta' \subseteq \Delta$

$\Gamma; \Delta, \Delta_O \setminus \Delta', \Delta_O; \Omega \Longrightarrow_1 \; A \multimap B \Uparrow$ By rule $\multimap R_v$

Case $foc - \&\mathbf{R}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma; \Delta; \Omega \Longrightarrow A \Uparrow} \quad \overset{\mathfrak{F}_2}{\Gamma; \Delta; \Omega \Longrightarrow B \Uparrow}}{\Gamma; \Delta; \Omega \Longrightarrow A \& B \Uparrow} \; foc - \& R$$

By i.h. on $\mathfrak{F}_i$, either $\Gamma; \Delta, \Delta_O \setminus \Delta_O; \Omega \Longrightarrow_0 \; G_i \Uparrow$

or $\Gamma; \Delta, \Delta_O \setminus \Delta_i', \Delta_O; \Omega \Longrightarrow_1 \; G_i \Uparrow \; \forall \; .\Delta_O$ and for some $\Delta_i' \subseteq \Delta$ for i = 1,2.

There are four $T - flag$ possibilities.

Subcase (0,0):
$\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A \Uparrow$ and $\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ B \Uparrow$.
Combining them by rule $\&R_{00}$ satisfies .
Subcase (0,1):
$\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ A \Uparrow$ and $\Gamma;\ \Delta, \Delta_O \setminus \Delta'_2, \Delta_O;\ \Omega \Longrightarrow_1\ \ B \Uparrow$.
Combining them by rule $\&R_{01}$ satisfies .
Subcase (1,0): Similar with the subcase (0,1) .
Subcase (1,1):
$\Gamma;\ \Delta, \Delta_O \setminus \Delta'_1, \Delta_O;\ \Omega \Longrightarrow_1\ \ A \Uparrow$ and $\Gamma;\ \Delta, \Delta_O \setminus \Delta'_2, \Delta_O;\ \Omega \Longrightarrow_1\ \ B \Uparrow$.
By rule $\&R_{11}$, $\Gamma;\ \Delta, \Delta_O \setminus (\Delta'_1, \Delta_O) \cap (\Delta'_2, \Delta_O);\ \Omega \Longrightarrow_1\ \ A\&B \Uparrow$
is desired result if we take $\Delta' = (\Delta'_1 \cap \Delta'_2)$.

Case $foc - \oplus\mathbf{L}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta;\ \Omega, A \Uparrow \Longrightarrow C} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta;\ \Omega, B \Uparrow \Longrightarrow C}}{\Gamma;\ \Delta;\ \Omega, A \oplus B \Uparrow \Longrightarrow C}\ foc - \oplus L$$

By i.h. on $\mathfrak{F}_i$, either $\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega \Longrightarrow_0\ \ G_i \Uparrow$

or $\Gamma;\ \Delta, \Delta_O \setminus \Delta'_i \Delta_O;\ \Omega \Longrightarrow_1\ \ G_i \Uparrow \forall\ .\Delta_O$ and for some $\Delta'_i \subseteq \Delta$ for i = 1,2 .

There are four $T - flag$ possibilities.
Subcase (0,0):
$\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ \ C$ and $\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_0\ \ C$.
Combining them by rule $\oplus L_{00}$ satisfies .
Subcase (0,1):
$\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_0\ \ C$ and $\Gamma;\ \Delta, \Delta_O \setminus \Delta'_2, \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_1\ \ C$.
Combining them by rule $\oplus L_{01}$ satisfies .
Subcase (1,0): Similar with the subcase (0,1) .
Subcase (1,1):
$\Gamma;\ \Delta, \Delta_O \setminus \Delta'_1, \Delta_O;\ \Omega, A \Uparrow \Longrightarrow_1\ \ C$ and $\Gamma;\ \Delta, \Delta_O \setminus \Delta'_2, \Delta_O;\ \Omega, B \Uparrow \Longrightarrow_1\ \ C$.
By rule $\oplus L_{11}$, $\Gamma;\ \Delta, \Delta_O \setminus (\Delta'_1, \Delta_O) \cap (\Delta'_2, \Delta_O);\ \Omega, A \oplus B \Uparrow \Longrightarrow_1\ \ C$
is desired result if we take $\Delta' = (\Delta'_1 \cap \Delta'_2)$.

Case $foc - \mathbf{decideL}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta;\ A \Downarrow \Longrightarrow C}}{\Gamma;\ \Delta, A;\ \cdot \Uparrow \Longrightarrow C}\ foc - decideL$$

Either

$\Gamma;\ \Delta, \Delta_O \setminus \Delta_O;\ A \Downarrow \Longrightarrow_0\ C$	By i.h. on F1 $\forall\ .\Delta_O$
$\Gamma;\ \Delta, \Delta_O, A \setminus \Delta_O;\ \cdot \Uparrow \Longrightarrow_0\ C$	By rule $decideL$

or

$\Gamma;\ \Delta, \Delta_O \setminus \Delta', \Delta_O;\ A \Downarrow \Longrightarrow_1\ C$	By i.h. on F1 $\forall\ .\Delta_O$ and for some $\Delta' \subseteq \Delta$
$\Gamma;\ \Delta, \Delta_O, A \setminus \Delta', \Delta_O;\ \cdot \Uparrow \Longrightarrow_1\ C$	By rule $decideL$

Goal:

$\Gamma;\ \Delta, \Delta_O, A \setminus \Delta'', \Delta_O;\ \cdot \Uparrow \Longrightarrow_1\ C$ $\quad \forall\ .\Delta_O$ and for some $\Delta'' \subseteq (\Delta, A)$; take $\Delta'' = \Delta'$ from above .

Case $foc - \otimes\mathbf{R}$: Suppose that the given derivation ends with the rule

$$\frac{\overset{\mathfrak{F}_1}{\Gamma;\ \Delta_1;\ \cdot \Longrightarrow A \Downarrow} \quad \overset{\mathfrak{F}_2}{\Gamma;\ \Delta_2;\ \cdot \Longrightarrow B \Downarrow}}{\Gamma;\ \Delta_1, \Delta_2;\ \cdot \Longrightarrow A \otimes B \Downarrow}\ foc - \otimes R$$

There are four $T - flag$ possibilities.

Subcase (0,0):

$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta_2, \Delta_O;\ \cdot \Longrightarrow_0\ A \Downarrow$	By i.h. on $\mathfrak{F}_1$, $\forall\ .\Delta_O$
$\Gamma;\ \Delta_2, \Delta_O \setminus \Delta_O;\ \cdot \Longrightarrow_0\ B \Downarrow$	By i.h. on $\mathfrak{F}_2$, $\forall\ .\Delta_O$
$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta_O;\ \cdot \Longrightarrow_0\ A \otimes B \Downarrow$	By rule $\otimes R_{00}$

Subcase (0,1):

$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta_2, \Delta_O;\ \cdot \Longrightarrow_0\ A \Downarrow$	By i.h. on $\mathfrak{F}_1$, $\forall\ .\Delta_O$
$\Gamma;\ \Delta_2, \Delta_O \setminus \Delta_O, \Delta';\ \cdot \Longrightarrow_1\ B \Downarrow$	By i.h. on $\mathfrak{F}_2$, $\forall\ .\Delta_O$ and for some $\Delta' \subseteq \Delta_2$
$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta_O, \Delta';\ \cdot \Longrightarrow_1\ A \otimes B \Downarrow$	By rule $\otimes R$

Goal:

$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta'', \Delta_O;\ \cdot \Longrightarrow_1\ A \otimes B \Downarrow$ $\quad \forall\ .\Delta_O$, $\Delta'' \subseteq (\Delta_1, \Delta_2)$; take $\Delta'' = \Delta'$ from above.

Subcase (1,0): Similar with the subcase (0,1).

Subcase (1,1):

$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta_2, \Delta_O, \Delta_1';\ \cdot \Longrightarrow_1\ A \Downarrow$	By i.h. on $\mathfrak{F}_1$, $\forall\ .\Delta_O$ and for some $\Delta_1' \subseteq \Delta_1$
$\Gamma;\ \Delta_2, \Delta_O, \Delta_1' \setminus \Delta_O, \Delta_2';\ \cdot \Longrightarrow_1\ B \Downarrow$	By i.h. on $\mathfrak{F}_2$, $\forall\ .\Delta_O$ and for some $\Delta_2' \subseteq (\Delta_2, \Delta_1')$
$\Gamma;\ \Delta_1, \Delta_2, \Delta_O \setminus \Delta_O, \Delta_2';\ \cdot \Longrightarrow_1\ A \otimes B \Downarrow$	By rule $\otimes R$ $\forall\ .\Delta_O$ and for some $\Delta_2' \subseteq (\Delta_1, \Delta_2)$

□