TÉCNICO LISBOA

# P-Res Tutor: An Intelligent Tutoring System for Propositional Resolution

**Alexandre Lopes Poeira**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisor: Prof. Manuel Fernando Cabido Peres Lopes

### Examination Committee

Chairperson: Prof. David Manuel Martins de Matos
Supervisor: Prof. Manuel Fernando Cabido Peres Lopes
Member of the Committee: Prof. Francisco António Chaves Saraiva de Melo

**October 2021**

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

# Acknowledgments

I would like to thank my parents and brothers for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my grandparents, aunts, uncles and cousins for their understanding and support throughout all these years.

I would also like to acknowledge my dissertation supervisors Prof. Manuel Lopes of Instituto Superior Técnico and Prof. Tatsuya Nomura of Ryukoku University for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

# Abstract

Intelligent Tutoring Systems (ITSs) are computer-based instructional systems that specify what to teach and how to teach it [1]. Given a specific knowledge base, an ITS presents activities related to this base for the user so they can ”learn by doing” in realistic and meaningful contexts, thus providing a personalised learning experience dependent on the user's experience. Constructing an ITS is a challenge in itself [2], being a multidisciplinary task involving multiple research fields, from which artificial intelligence and education stand out. To simplify, the ”intelligent” part of an ITS can be independent of the knowledge base, being instead based on the empirical estimation of learning progress of the user given the correctness of their answer [3].

This dissertation focuses on the construction of an online ITS, P-res Tutor, with the purpose of teaching Propositional Resolution [4], a rule of inference of Propositional Logic, and its applications in solving theorems. Since learning this rule requires learning other subjects, the system also teaches the basics of Propositional Logic and the Conjunctive Normal Form (CNF) with different activities based on Logic exercises and other rules of inference. For this system, we apply multi-armed bandit methods [3] and develop a unique error diagnosis process [5] for exercise selection and student modelling.

The prototype of the system was implemented and a user-study was conducted from which we got results validating our system's teaching potential.

# Keywords

Intelligent Tutoring Systems; Propositional Logic; Propositional Resolution; Multi-Armed Bandits;

# Resumo

Os Sistemas Tutoriais Inteligentes (STIs) são sistemas de aprendizagem baseados em computador que especificam o que ensinar e como ensinar [1]. Dada uma base de conhecimento específica, um STI apresenta atividades relacionadas a essa base para que o utilizador possa ”aprender ao trabalhar” em contextos realistas e significantes, proporcionando assim uma experiência de ”aprendizagem personalizada” dependente da experiência do utilizador. Construir um STI é um desafio em si [2], sendo uma tarefa multidisciplinar que envolve múltiplos campos de pesquisa, dos quais se destacam a inteligência artificial e a educação. Para simplificar, a parte ”inteligente” de um STI pode ser independente da base de conhecimento, sendo fundamentada na estimativa empírica do progresso de aprendizagem do utilizador dado a sua resposta [3].

Esta tese foca-se na construção de um STI online, P-res Tutor, com o objetivo de ensinar a Regra da Resolução [4], uma regra de inferência da Lógica Proposicional, e as suas aplicações na resolução de teoremas. Visto que aprender esta regra requer adquirir outros conhecimentos, o sistema também ensina os fundamentos da Lógica Proposicional e da Forma Normal Conjuntiva (FNC) com diferentes atividades baseadas em exercícios de Lógica e outras regras de inferência. Para este sistema, aplicamos métodos de Multi-Armed Bandits [3] e desenvolvemos um processo único de diagnóstico de erros [5] para seleção de exercícios e criação de um modelo de estudante. O protótipo do sistema foi implementado e um estudo de utilizadores foi conduzido, do qual obtivemos resultados que validam o potencial de ensino do nosso sistema.

# Palavras Chave

Sistemas Tutoriais Inteligentes; Lógica Proposicional; Regra da Resolução; Multi-Armed Bandits;

# Contents

# List of Figures

# Acronyms

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **CNF** | Conjunctive Normal Form |
| **KCs** | Knowledge Components |
| **MAB** | Multi-Armed Bandits |
| **RiaRit** | Right activity at the Right time |
| **ZPD** | Zone of Proximal Development |
| **ZPDES** | Zone of Proximal Development and Empirical Success |

# 1

# Introduction

## Contents

## 1.1 Motivation

As a teaching supplement, computer courses have been researched and developed intensively. Inevitably, Artificial Intelligence (AI) methods were introduced to the area of online teaching, from which the area of *Intelligent Tutoring Systems* was born. These systems, in a general manner, are computer courses which change according to the student's input [6], providing an individualised and personalised learning experience. For a student to learn a skill, a sequence of exercises or activities is followed. For an expert/teacher, the challenge lies on creating a sequence that maximizes learning for every student. An ITS serves as an artificial expert that determines the optimal sequence of activities for the acquisition of the skill, while being customized to fit each student's unique characteristics.

The design and development of ITSs require resources from multiple research fields, including artificial intelligence, cognitive sciences, education, human-computer interaction, and software engineering, all of this without taking into account knowledge about the ITS's main subject of teaching. As such, building this kind of system presents a thoroughly challenging task, given the massive multidisciplinary requirements [2].

There are two main motivating factors for researchers to build an ITS [7]:

- *Pure Research Needs*: Since ITS research lies at the intersection of multiple domains, an ITS can provide an excellent test-bed for various theories from cognitive psychologists, educational theorists and/or AI scientists. Different teaching scenarios can be simulated by changing parameters of the system, such as measuring the impact of different activities being presented to the user.
- *Practical Needs*: Since generally there are more students than teachers, most educational systems become geared towards group teaching methods, while losing most advantages that are found in one-on-one tutoring. One of the primary advantages of an ITS is the possibility to provide one-on-one tutoring without necessarily losing the advantages of the group teaching environment, for example by providing each student their own copy of the ITS.

The aim of this thesis is, thus, to create a practical system which can serve as a research tool for ITS researchers and as a teaching tool for anyone who wishes to teach or learn Propositional Resolution.

## 1.2 Overview

When constructing an ITS, one must consider its functions and its structure, as both can be wildly different depending on the author [2]. For the proposed system, it provides the functions of an exercise generator with three different types of activities, a simplified method of knowledge estimation and student model construction utilizing Knowledge Components (KCs) and the Zone of Proximal Development (ZPD), and a Multi-Armed Bandits (MAB) based algorithm to choose activities for the user to

perform.

The proposed system includes three activities which verify the knowledge needed to learn all propositional logic rules of inference, *Truth or False*, Conjunctive Normal Form (CNF) conversion and *Applying the Resolution Principle*. Connecting all of these activities is the overarching activity of a *Full Resolution exercise*, which combines all of the previous activities into a multi-step exercise that teaches the *Resolution* rule of inference.

Logic is considered to be an important subject for students of computer science [8], helping them in developing their skills of logical analysis, problem resolution and formalization. These skills, in turn, are necessary for other computing related activities such as programming, specification of requisites, database development, within others. Despite being simple in theory, mastery of the basic concepts of propositional logic requires practice, which, in essence, the developed ITS provides as it serves as a training grounds for users to learn and practice concepts of propositional logic.

Propositional Logic is a great field of study for an ITS [9] because of its inherent simplicity, since it allows us to easily split it into simpler skills, KCs, that the user has to learn, and place them in an hierarchy of complexity so as to know in what order each skill must be learned to achieve better learning. Another advantage is the ease of exercise generation, given its limited symbols but its freedom in their use when generating logical sentences, it becomes straightforward to generate random yet relevant exercises.

The exercise selection will be determined by MAB methods already proven effective in other systems [3, 10] integrated with an error diagnosis process original to this system and inspired by the system IATS [5]. This process was introduced to figure out the relevance of different types of errors in the student's learning, by influencing the exercise selection depending on what errors the user makes.

## 1.3 Objectives

The main goal of the thesis, as explained before, is to construct an ITS with the purpose of teaching Propositional Resolution. This system must include exercises and activities that test every skill needed to learn Resolution. Subsequently, the system must also estimate the knowledge of the student given their answers on the exercises shown and select exercises accordingly using this student model.

This system must also provide feedback to the learner, so they can operate independently, and to the teacher, so the system can be improved if needed and to study the importance of different adjustable parameters in the system. These can range from having a limited set of fixed exercises to adjusting the learning speed.

Finally, our overall goal with this system is to test the selection of exercises given the user's input, by integrating the MAB methods with the error diagnosis process.

## 1.4 Following Chapters

The remainder of the document is split in 4 parts. Chapter 2 is comprised of the related work and state-of-the-art analysis of the domain of the thesis. ITS structure, construction and authoring tools are discussed, as well as a quick revision on logic rules of inference needed to complete the ITS's exercises. Chapter 3 is where the system construction and implementation is described, including the algorithm used for exercise selection, the used libraries from the book *AIMA* [4] and the needed changes to them, and the student modelling, including the defined KCs and the ZPD.

Chapter 4 is where we detail the user study and comment on its results. Finally, Chapter 5 is composed of the conclusion of the paper, where we review what we have accomplished with this dissertation and future work to be done to improve the system further.

# 2

# Related Work

## Contents

## 2.1 ITS Construction

Due to their complexity, ITSs are normally not easy to construct [2]. Since the resources needed come from very different areas, such as artificial intelligence, education, human-computer interaction and software engineering, the process of building an ITS is inherently a challenging task. The construction also highly depends on the views and priorities of the author. They can choose to focus on exercise selection [3, 11], ensuring the system's tutoring decision making is based on sound pedagogical principles, or in error diagnosis [12, 13], using appropriate knowledge structure and algorithms to interpret users' decisions correctly and give proper feedback to the user.

### 2.1.1 ITS Structure and components

The structure of an ITS varies extremely between different systems. Since the work in this area is experimental in nature, there is no clear-cut way to construct an ITS's structure [7] although many common elements can be found in the structure of different ITSs. The most common structure is a composition of different modules comprising different parts needed for the tutoring process. The basic components, *Expert Knowledge module*, *Student Model module* and the *Tutoring module*, make up the intelligent part of the system and is where most work in constructing an ITS is had, while a fourth component, the *User Interface module*, has also been identified [14] as essential as it serves as the only communication channel between the user and the system. A simplified model of this structure can be found in the following figure 2.1.

The Expert Knowledge module, or domain expert, comprises of all the rules and facts of the domain to be conveyed to the student by the ITS. It serves as the source of knowledge to be presented to the student, including generating questions, explanations and answers, and as a standard for evaluating the student's performance. For this last task, the module must be able to generate solutions to problems in the same context as the student, including not only surface knowledge, the meaning and descriptions of the different concepts the student must learn, but also the ability to interpret, understand and solve the same exercises it presents [14].

The Student Model module is a representation of the student's knowledge and skills, using knowledge attained by analyzing the input of the user. It includes a clear evaluation of the skills of the student, to be compared with the Expert Knowledge model and better ascertain their overall mastery of the ITS's domain. Ideally, this model should include all aspects of the student's behaviour and knowledge that have possible repercussions on his or her performance. However, the task of constructing such a model is normally impossible, as the only mode for communication with the ITS, normally a keyboard and mouse, is too restrictive and cannot communicate factors that can influence the student's performance and knowledge, such as the student's state of mind. As such, other approaches with a simpler Student

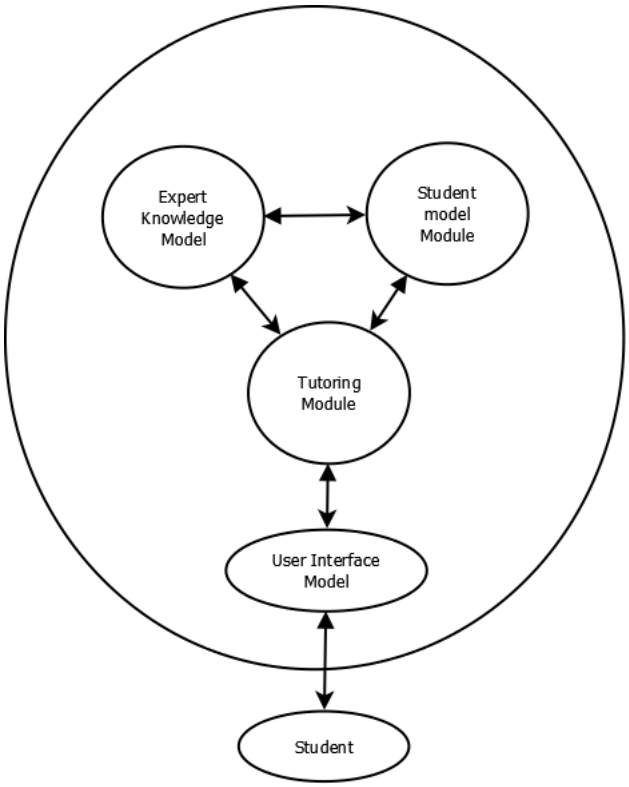


**Figure 2.1:** Basic structure of an ITS

Model have been researched [3] [10].

The Tutoring module is responsible for the teaching strategy, working with information from the Student Model to decide which activities or information to present to the student, such as hints to overcome impasses in performance, advice, support, explanations, different practice tasks, tests to confirm hypotheses in the student's model, etc. How much influence the tutoring module has is entirely up to the ITS creators, ranging from systems that give almost full control of the activity to the student, to systems that monitor the student's every activity very closely, adapting their actions to the student's responses. In summary, this module is the sole source and orchestrator of any pedagogical decision made by the system [14].

Finally, the User Interface module controls the interactions between the system and the learner. Normally ITSs tend to use elaborate graphic interfaces, in order to make interaction with the system more user-friendly. Because the user interface can ”make or break” the ITS, no matter how intelligent the other components are, it cannot be considered a secondary component to the other modules, as it is the final form in which the ITS presents itself to the user. Most work done on this module falls under the realm of interface designing [14].

### 2.1.2 Common approaches when constructing an ITS

Regarding ITS construction [2], there are two main approaches that utilize help from pre-existing systems: the shell-based approach and usage of ITS authoring tools.

The shell-based approach is well-known in artificial intelligence. It uses a shell, a software development environment containing basic elements to construct different computer programs, as the basis for its structure. Shells provide code libraries and conceptual frameworks to build parts of the ITS. The main advantage of this approach is that it allows the creator to better customize and adapt the new system to the domain to be taught, although it requires the creator to have programming skills to do so. Overall, shells tend to focus on the bigger picture, allowing creators to more easily create a functioning ITS, while approaches that target specific ITS components are more profound and detailed. Some of the more better-known shells include E-Mycin [15] and, more recently, the Tutor-Expert System [16].

For a creator with no programming skills, the better approach is using ITS authoring tools, which go beyond the simple shell and provide an additional user-interface for non-programmers to formalize and visualize their knowledge, thus simplifying the process of authoring an ITS [17]. There are various different tools for different purposes, they can serve to sequence the teaching of progressively complex content, provide a learning environment for students to learn skills by practicing them and receive feedback or to simply assist ITS authors in designing and constructing their system. Some of the better known tools include RIDES (for Rapid ITS Development Environment) [18] and REDEEM [19].

Overall, there are some principles to consider when building an ITS: Approaches involving small building blocks should be preferred to reduce the time to produce a functional ITS; and increased assistance to authors is required, since currently available tools and shells are still too complex for non-programmers [2].

## 2.2 Knowledge Estimation

For the purpose of teaching a student, the system must also ascertain what the student already knows, building the student model, but also of what he/she must know to achieve the taught skill, by having a clearly defined expert knowledge module.

### 2.2.1 Knowledge Components

In any Intelligent Tutoring System, we can define its main goal as the learning of a certain skill, in this case being Propositional Resolution. To facilitate the learning process, this goal can be subdivided into sub-goals, smaller skills that must be learnt first since they are applied in the main skill. These sub-goals are called Knowledge Components, KCs. They can be represented as nodes in an acyclic graph to allow

the creation of a hierarchy between them to model their increasing difficulty and required precedence over other KCs.

One thing to note is that there is normally no direct relation between KCs and learning activities, meaning each activity can provide an opportunity to acquire different KCs instead of having a different activity for each one. This is considered when defining which activities are to be performed in the ITS, and when estimating the student's knowledge and acquisition of KCs.

As defined by Lindsey et al. [20], KCs are provided by experts on the matter which can then be refined by automated techniques. For this problem, we will propose our own KCs required to learn Propositional Resolution, which we will discuss further in this dissertation.

### 2.2.2 Zone of Proximal Development

When learning a skill, it is important to take note of what activities to perform. If they are too easy, no new knowledge is attained, if they are too hard, the student cannot complete or understand them, meaning that the exercises recommended can neither be too hard nor too easy. They must be challenging enough to maintain the student's interest and easy enough that they can solve them, which requires exercises to be at a level of required knowledge only slightly higher than the current. These activities belong to what is called the Zone of Proximal Development [21], or ZPD, for short, which comprises of all the activities which give the most learning progress.

## 2.3 Multi-armed bandits for ITS

When making an ITS specialized in choosing sequences of exercises customized for each student, the most pressing issue is how to define and optimize this sequence. A solution is using MAB methods [22]. In short, in a problem with various activities to perform and each with a different unknown reward, the problem lies in choosing whether to explore different activities to see the reward of each one, "Exploration", or exploit the activity that is guaranteed to give a good reward, "Exploitation". Taking a casino analogy into account, multi-armed bandits would be described as trying to find the most profitable slot machine.

Since the agent, or bandit in the casino analogy, must spend money exploring all of them before choosing the best one, the dilemma lies in simultaneously trying new activities to know their payoff while also selecting the best ones so actual profit can be made. In the context of an ITS, the slot machines are the different exercises that the algorithm recommends to the student, the reward is the learning gain and the choice of "Exploration vs Exploitation" is made by the teaching algorithm choosing a type of exercise known to provide learning and selecting different activities with an unknown reward, considering the reward information it gets previously.

The usage of these methods allows us to have a weaker dependency on a Student Model, in favor of optimizing the Tutoring module to adapt to each individual student. This, in turn, also allows us to use methods of student model construction that make no assumptions about how students learn and only require information regarding the estimated learning progress of activities, which creates a simple, yet unique model for each user. As such, by focusing on optimization of the MAB algorithm, the system becomes more accurate as student models and the tutoring module become more defined.

There are some particularities in an MAB approach to ITSs. Firstly, the reward for each activity, which is learning progress, does not stay the same, since it depends on the competence level of the student for the exercise, which will stop giving a reward after a certain competence level is achieved. Secondly, rewards are not independent and identically distributed, as we are dealing with humans, which brings various effects into play that can affect the reward, such as distractions, mistakes using the system and mainly different preferences between students.

Thus, every activity $a$ will have a weight $w_a$ which tracks its reward, correlated to the learning progress given by activity $a$. Each time this activity is performed, this $w_a$ is updated related to the reward and the previous weight:

$$w_a \leftarrow Bw_a + ur \tag{2.1}$$

As shown, the reward given by the activity, $r$, is added to the current weight of the activity to update the new weight. The main advantage of this update is its simplicity, allowing it to be used with a single variable, the reward. By altering parameters $B$,$u$, we can change the relevance of the reward or the previous weight. These weights come into use when the system has to choose the next activity to recommend, as each activity is assigned a probability, $p_i$, for it to be selected, which uses the normalized weight of the activity, $\acute{w}_a$, the exploration rate, $y$, and a uniform distribution, $e_u$, to ensure sufficient exploration of activities. This probability is calculated in the following way:

$$p_i = \acute{w}_a(1-y) + ye_u \tag{2.2}$$

Since it is needed that all activities have an associated weight to determine their probability, it would be needed to explore all activities to estimate their impact on each knowledge component. This would be very time-consuming and could produce an under-performing learning sequence, so instead, a canonical learning sequence is used to initialize the algorithm, after which this sequence can be optimized. This sequence is determined by an expert, and normally has a higher reward on the introductory low difficulty activities and a lower reward in the more advanced activities.

Two different algorithms using MAB technology can be used, the first one requiring little domain knowledge called Zone of Proximal Development and Empirical Success (ZPDES). The second approach assumes there is a simple relation between the activities and skills of the student, estimates the

learning progress obtained at a given point in time and proposes to the student the activities which provide higher learning progress, thus the name of the algorithm being Right activity at the Right time (RiaRit) [3].

### 2.3.1 ZPDES

This algorithm is inspired by the ZPD and the empirical estimation of learning progress, and, as such, it requires very little domain/user knowledge. To estimate the reward of each skill, that is, the estimated learning that the skill provides, the correctness of the answer given by the student is the only parameter needed.

Instead of comparing the correctness of the answer given at a certain time $t$ with all of the previous answers $d$, it instead compares the last half of $d/2$ answers with the earlier half of $d/2$ answers given. This allows the measure of the quality of each activity, since we can measure how much progress a certain activity has provided in a short time window and we consider activities with a faster progress to be better than others with a slower progress [23].

Thus, the computation of the learning progress $r$, where $C_k = 1$ if the exercise at time $k$ is correct, is as follows:

$$r = \sum_{k=t-d/2}^{t} \frac{C_k}{d/2} - \sum_{k=t-d}^{t-d/2} \frac{C_k}{d/2} \tag{2.3}$$

When an activity has already been acquired or when the student is not progressing in any way, which are both extreme cases, the reward given will be zero. To reduce the number of activities that are needed to explore, there is also the added restriction that only activities in the ZPD are selected for the user.

Initially, the ZPD is defined as a graph with every activity ordered by levels of difficulty. Only the most basic skills are included in the ZPD, with more advanced ones having the prerequisite of attaining previous easier skills. For activities already in the ZPD, free exploration is allowed since these are considered to always give some learning progress. After enough progress is attained in a certain activity, it is considered mastered, is removed from the ZPD and the more advanced skills that had the previous one as prerequisite being added to the ZPD.

This algorithm also allows the tutor to limit or expand exploration of activities if needed, depending on whether the set of activities have a clear progression of difficulty between them. If they do, then exploration is limited to force students to follow a specific path between each skill, if not, meaning different students can have very different orders when they are obtaining skills, then wider exploration is allowed in order to accommodate individual differences.

### 2.3.2 RiaRit

Another algorithm based on the ZPD but using more knowledge about the domain and the student, is the RiaRit algorithm. For this approach, the competence level of the student of each KC is defined as a continuous number between 0 and 1, with 0 meaning not acquired and 1 being fully acquired, with $c_i$ being the competence level for the knowledge component $KC_i$. Then, for each activity $a$ and each knowledge component $KC_i$, a competence value $q_i(a)$ is defined by an expert as the level of competence required for each $KC_i$ to have maximal success in activity $a$, meaning that, when this level of competence is achieved, everything activity $a$ can contribute in terms of learning progress towards acquiring $KC_i$ has been learned by the student.

These competence values are used to estimate the impact of each activity in the student's overall learning progress. As such, this estimation of competence values is a high priority in this approach and is done by relying on a simplified version of Knowledge Tracing [24] based on the relation between the activities and the KCs. Considering a knowledge component $i$ for which the student has a competence level $c_i$, when the student succeeds in performing action $a$ and $c_i < q_i(a)$, then the student's competence $c_i$ is being underestimated and should be increased. In the opposite case, when the student fails the action $a$ and $c_i < q_i(a)$, then the student's competence $c_i$ is being overestimated and should be decreased. The reward for this method can thus be defined as the difference between $q_i(a)$ and $c_i$:

$$r_i = q_i(a) - c_i \tag{2.4}$$

With this reward, the competence values are updated after each activity, with the addition of the parameter $f$ to adjust the confidence we have of the reward:

$$c_i = c_i + fr_i \tag{2.5}$$

Then, an expert is asked to determine minimum competence levels $m_i(a)$ for each activity a, for them to be unlocked for exploration. If the competence level $c_i$ of activity $a$ in the ZPD is higher than $m_i(a)$, then a is allowed free exploration. Progress through the ZPD is mostly the same as the ZPDES approach but with the added attribute of competence levels and is as follows: When a minimum competence level of a $KC_i$ is achieved, the activities which influence this $KC_i$ are unlocked and allowed exploration if the actions preceding it are already all unlocked. At the same time, when the competence level of all $KC_i$'s of action $a$ reach $m_i(a)$, this activity is no longer explored, since it is considered it no longer has anything to teach.

## 2.4 Error diagnosis

For the purpose of constructing and maintaining a stable student model, analysing the user's input in depth can contribute significantly to this effort. An ITS can be said to consist of two different loops, an outer loop, which chooses activities for the student by matching their learning progress to adequate activities, and an inner loop, which gives feedback and hints about steps the student must take to solve the activity [25]. An important responsibility of the inner loop is analyzing these steps the student takes, in order to find exactly where the user made a mistake, and give proper feedback.

Different approaches for error diagnosis have been studied extensively in ITSs [26] and 8 different aspects have been identified in multiple ITSs as relevant for diagnosing student steps:

- **Correctness**: refers to whether or not a student step matches an expected step, with the only possible outcomes being *correct* or *incorrect*;
- **Difference**: similar to correctness but measures the edit distance, how different it is, between the student step and the expected step. This measure is normally a number or percentage, for example, if the only difference between the student step and the correct step is a single character, a '+' for a '-', then the edit distance would be one, since it requires only one edit operation for the student step to be correct;
- **Redundancy**: refers to whether the student step is significant in any way, if the difference between the current student step and the previous one is too small, it can be considered redundant. Possible outcomes are *redundant*, *not redundant* and *unknown*;
- **Type of Error**: refers to classifying errors, for example, classifying $a+(b$ as a syntax error. Possible outcomes depend on the domain of the ITS;
- **Common Errors**: refers to errors students make based on common misconceptions, for example, when forgetting to change the sign when moving an expression to another side of an equation ($a + b = 0 \rightarrow a = b$). Possible outcomes depend on the domain of the ITS;
- **Order**: refers to the order in which the student takes different steps, with the possible outcomes being *correct order*, *incorrect order* and *unknown*;
- **Preference**: refers to existing preferable solutions to problems than the one the student presents, with the possible outcomes being *preferred*, *not preferred* and *unknown*.
- **Time**: refers to the time the student took to submit a step or solve a problem, normally measured in milliseconds.

Of these 8 aspects, correctness is the most common aspect, as it is used in most if not all ITSs, and many other aspects rely on it. For example, type of error and common error can only be found in student steps that are deemed incorrect and preference can only be determined between correct steps. While many systems measure time, only a few use it for diagnostic purposes. Since most ITSs can be accessed at home without supervision, it becomes difficult to monitor how much time is actually spent on a question, for example, a student might take a break while in the middle of an activity. This makes it impractical for diagnosis purposes.

In regards to diagnostic processes, most ITSs use multiple aspects for diagnosing student responses. The most basic process consists of a single aspect, correctness, only checking whether the answer is correct or not. A more complex process involving more aspects is used in the system *AITS* [5] in the figure 2.2. By calculating the edit distance between the student and the ideal step, other aspects of the error can be better analyzed by checking the number and the content of the different nodes to determine redundancy and the type of error. Using these aspects, an error is classified according to its completeness and its accuracy, meaning an incorrect student step can be either *complete but inaccurate*, *incomplete but accurate* and *incomplete and inaccurate*. The diagnosis *complete and accurate* never occurs since it means the edit distance is zero, and the student step is equal to the best response.

## 2.5 Propositional Logic

Propositional Logic [4] is a branch of logic that deals only with propositions, which are facts represented by symbols. They can be a single affirmation or literal, which is called an atomic proposition (an example of this is the proposition “It is raining”), or a set of these affirmations/literals connected by logical connectives, which is called a compound proposition (an example of this is “It is raining, and it is cloudy” which is a compound proposition of the literals “It is raining” and “It is cloudy” with the logical connective “and”).

Propositions are normally represented by uppercase letters (we can represent the propositions “It is raining” and “It is cloudy” as $A$ and $B$, respectively, and “It is raining, and it is cloudy” as “$A \wedge B$”). Unlike first-order logic, propositional logic does not deal with non-logical objects, such as predicates, or quantifiers, such as $\forall$, the universal quantifier, and $\exists$, the existential quantifier. There are two proposition symbols with fixed meanings: True is the always-true proposition and False is the always-false proposition.

Logical connectives are symbols used to connect two literals to create a compound proposition, except for the negation symbol, which is the only connective that operates on a single proposition.
The logical connectives used in Propositional Logic are:

- **Negation**; represented by $\neg$, $\sim$, or “NOT”, is the only connective that is used in relation to a single

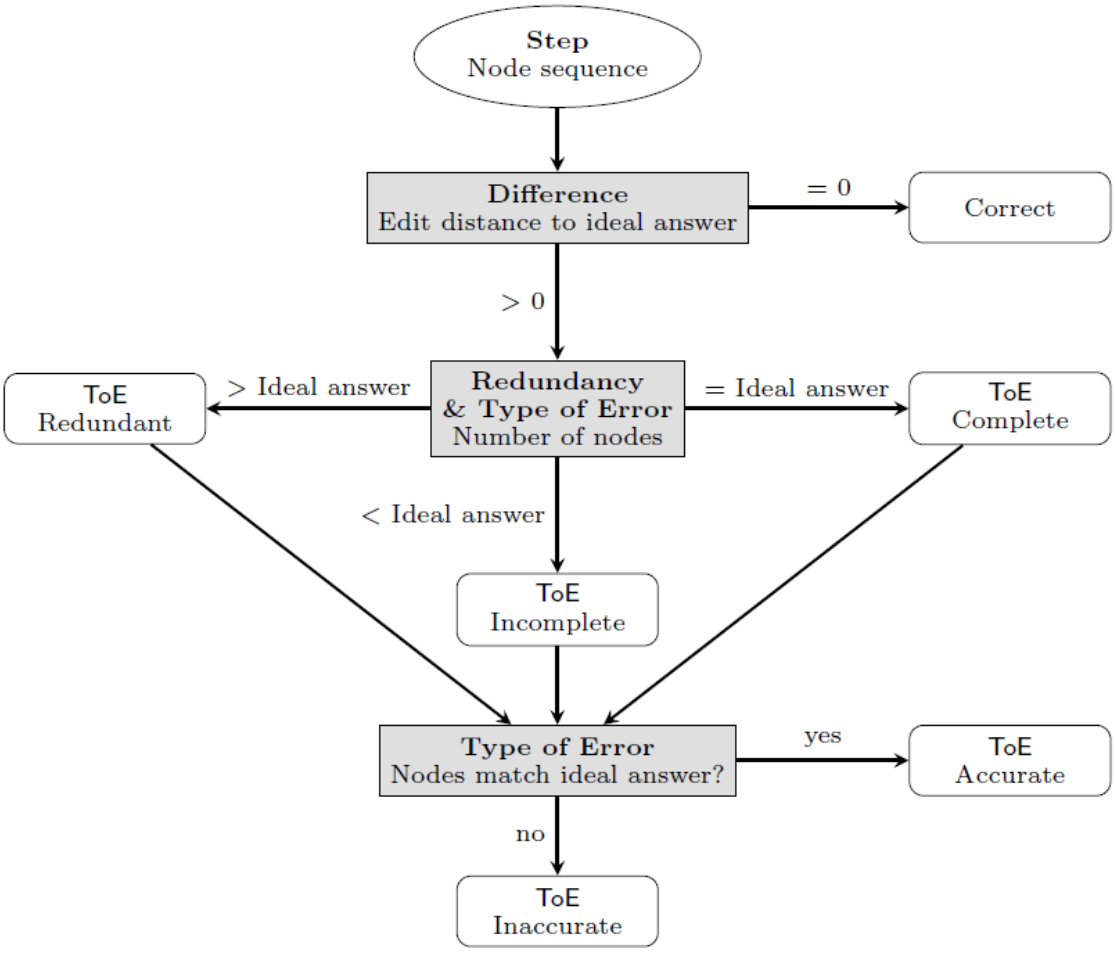


**Figure 2.2:** Error diagnosis process of *AITS* [5]

literal and isn't used to connect two literals. Represents denial of a literal, for example: $\sim A$ is only true when $A$ is false.

- **Conjunction**; represented by $\wedge$, &, $\cap$ or "AND". Represents the intersection between two literals, for example: $A \wedge B$ is only true when both $A$ and $B$ are true.

- **Disjunction**, represented by $\vee$, |, $\cup$ or "OR". Represents the junction of two literals, for example: $A \vee B$ is true when $A$ or $B$ or both are true.

- **Implication**, represented by $\implies$ , ==>, or "IF...THEN". Represents dependence between two literals, for example: $A \implies B$, meaning if $A$ happens then $B$ must happen, is true when $B$ is true or when $A$ is false.

### 2.5.1 Resolution

Resolution is a logical rule of inference that leads to a theorem-proving technique in propositional logic and first-order logic. The resolution rule states that, from two different propositions, a new one can be created by uniting both and removing the complementary literals. The new proposition is said to be the *resolvent* of the previous two. This can be seen in the following expression where $P_1$ and $P_2$ are propositions and $l \in P_1$ and $\neg l \in P_2$:

$$Res(P_1, P_2) = (P_1 - \{l\}) \vee (P_2 - \{\neg l\}) \tag{2.6}$$

A set of complementary literals is a set of a literal and its negation, for example $A$ and $\neg A$. The *resolvent* between these two literals is the empty set $\{\}$. If there are other literals in the proposition, the same principle is applied and the complementary literals are removed, for example with the propositions $P = A \vee B \vee \neg C$ and $Q = B \vee C \vee \neg E$, if we apply the resolution rule, we get the expression:

$$Res(P, Q) = ((A \vee B \vee \neg C) - \{\neg C\}) \vee ((B \vee C \vee \neg E) - \{C\})$$

$$Res(P, Q) = (A \vee B) \vee (B \vee \neg E)$$

$$Res(P, Q) = A \vee B \vee \neg E$$

The only restriction for this rule is propositional resolution can only be applied to propositions in the CNF or *clausal form*. A proposition is in the clausal form if it is a conjunction of one or more clauses, where a clause can either be a single literal, $A$ or $\neg B$, or a disjunction of literals, $A \vee B$ or $\neg C \vee D$. The only logical connectives a proposition in the CNF can contain are *AND*, *OR* and *NOT*, and the *NOT* operator can only be used in reference to a single literal. The empty set is also a clause, $\{\}$, and it is equivalent to an empty disjunction.

To convert sentences to the clausal form, the following steps must be taken:

1. Convert any implications to its disjunction form, Ex: $A \implies B$ is equivalent to $\neg A \vee B$;
2. Push the negation symbol inwards using De Morgan's Laws:
   (a) For double negation: $\neg\neg A$ is equivalent to $A$;
   (b) For negated disjunction: $\neg(A \vee B)$ is equivalent to $\neg A \wedge \neg B$;
   (c) For negated conjunction: $\neg(A \wedge B)$ is equivalent to $\neg A \vee \neg B$;
3. Apply the distributive property of disjunctions, Ex: $(A \wedge B) \vee C$ is $(A \vee C) \wedge (B \vee C)$.

An application of the resolution rule is to prove theorems. With the application of the satisfiability theorem, which dictates that given a set of premises $P$ and a literal $C$ we wish to prove, the premises logically

entail the literal $C$ if $P \wedge \neg\{C\}$ is unsatisfiable and vice-versa, we deny our theorem and attempt to reach the unsatisfiable state, the empty set $\{\}$, with the given premises. With the set of premises and a denied conclusion in the clausal form, we must reach the empty set by applying the resolution rule to prove the theorem. For example:

a

b

c

$$P = \{\{A\}, \{\neg A, B\}, \{\neg B, D\}\}$$

$$C = \{\{D\}\}$$

$$P \wedge \neg\{C\} = Q = \{\{A\}, \{\neg A, B\}, \{\neg B, D\}, \{\neg D\}\}$$

$$Res(Q_1, Q_2) = R_1 = \{B\}$$

$$Res(R_1, Q_3) = R_2 = \{D\}$$

$$Res(R_2, Q_4) = \{\}$$

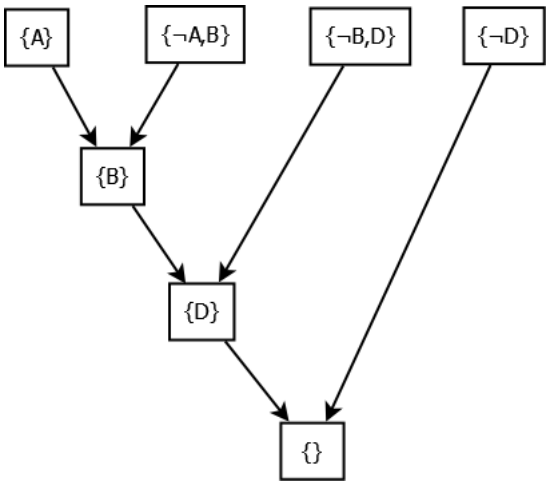


**Figure 2.3:** Resolution rule application to find a contradiction in a CNF proposition

Since we managed to reach the empty clause, $P \wedge \neg\{C\}$ is unsatisfiable and the literal $C$ is proved to be the conclusion of the premises $P$ by proof by contradiction.

This entire process of Propositional Resolution, from converting a logical sentence to its clausal form to achieving the unsatisfiable state with the premises and the denied conclusion, is the main subject we wish to teach with this ITS, which, by consequent, will also teach the basics and rules of inference of

Propositional Logic.

## 2.6 Other Logic ITSs

Various tutoring systems with the purpose of teaching logic have already been explored extensively. Most of these systems differ from others by presenting different activities or being about specific parts of the Logic subject matter, with the most popular subjects for systems being proof deduction and theorem proving.

In the realm of Propositional Logic, most systems deal with teaching natural deduction. *Heraclito* [27] is a system that shows students an interface where they can input external exercises, different logic propositions, and apply rules of inference to solve them. A mediator agent evaluates the student's progress by reacting to different inputs, or a lack of input, and presents different strategies to solve the exercise. It mainly serves as an environment for students to train their reasoning, while a virtual expert judges their performance.

*EvoLogic* [13] is an upgrade of the former system, having all the same features and an improvement in the construction of the solution. Instead of just creating one optimal solution, it utilizes a Genetic Algorithm as a specialist agent to create multiple ones, allowing the mediator agent to give feedback on different forms of reasoning.

Another system which influenced the previous ones is *P-Logic Tutor* [9]. This system's primary purpose is to teach students fundamental concepts of propositional logic and theorem-proving techniques. For this purpose, it is divided in three modules, where students can compose formulas, create a truth table and check its correctness, attempt to show how a conclusion can be derived from one or two axioms, attempt to prove theorems by incrementally applying rules of inference, etc. At any time, the student can open a help window to assist them on traversing the interface or on understanding the subject matter itself, either by looking up concepts of propositional logic or by requesting a hint in solving the current exercise.

It is worth noting that most of these systems focus on assisting the student in performing a preselected exercise sequence, but do not include any mechanism that chooses exercises specially tailored to each student. In this matter, the proposed system will differentiate itself from previously made systems, as one of its focus areas is in choosing the right exercises to show the student.

# 3

# Implementation

## Contents

## 3.1 Overview

P-res tutor is designed to teach students Propositional Logic, with a focus in teaching the rule of Resolution. It is composed of three different activities the user can choose from or a stream of activities chosen by the system using the ZPDES algorithm, where the possible activities are one of the three. The three activities are:

- Truth or False exercises, where a formula and its supposed CNF formula are shown, and the user must answer whether the CNF formula is correct or not, found in the figure A.1;
- CNF conversion, where a formula is shown and the user must input its clausal form, found in the figure A.2;
- Connecting Clauses: Application of the resolution rule, where a formula in its CNF form is shown and the user must figure out if it is possible to achieve the empty clause by applying the resolution rule and eliminating clauses. This activity consists of selecting clauses to apply the resolution rule, the activity ends when the user can achieve the empty clause with any combination of the existing clauses or if the user selects it is not possible to do so , found in the figure A.3;
- Full Resolution exercise, where a set of premises $P$ and a conclusion $C$ are presented and the user must prove that the conclusion is proven by the premises, by converting $P \wedge \neg C$ to its clausal form, and apply the resolution rule to the remaining clauses to achieve the empty clause $\{\}$. It is the only activity to consist of two different steps, one where the user must input the clausal form of $P \wedge \neg C$ and another one where the user must connect the resulting clauses to achieve the empty clause, found in the figure A.4.

The prototype of the system is built on a python Flask application in an internet browser form 3.1, with the user interface being a free *HTML5* template available online and the tutoring module, student model module and expert knowledge module built entirely with Python. For the propositional logic structures, operations and expressions, a python library *logic.py* from the book *AIMA* [4] was used. This library provides a framework for propositional logic in a python context, from which we use structures for logical expressions and operations with logical connectives, such as all the steps of conversion to the clausal form.

### 3.1.1 Interaction Loop

When choosing a specific activity function, a random exercise of the chosen type will appear. After this, the user receives feedback for the correctness of the input and can check the solution. When using the stream of exercises option, an exercise is selected based on the student model and presented.


**Figure 3.1:** User interface of P-res Tutor

After the student's input, the correctness of the exercise is evaluated by the expert knowledge model, the student model is updated according to the correctness, feedback is shown and a new exercise is selected and presented based on the new student model. The solution for the previous exercise can also be consulted. This sequence of modules in the stream of exercises can be seen in the following figure 3.2.

This stream of new exercises is only stopped if the student wishes to go to the main menu or if the student model shows that the user has learned everything there is to learn. If a user wishes to start over and reset the student model, there is an option that resets it to its initial state in the main menu 3.1.

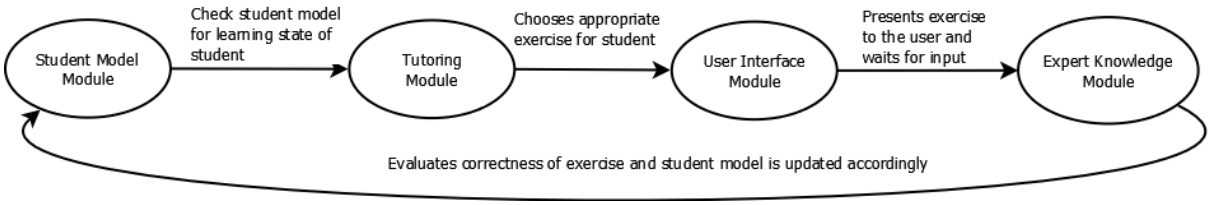


**Figure 3.2:** Diagram with the four modules of P-res Tutor and their purpose in the stream of exercises

## 3.2 Knowledge Estimation

To estimate the knowledge of the student regarding Propositional Resolution, we divide it into different Knowledge Components and represent them in the Student model module.

### 3.2.1 Knowledge Components

The proposed KCs for Propositional Logic and Propositional Resolution and its corresponding KC tree are the following:

- For the following KC examples we use the logical sentences $A =$"It is raining" and $B =$"It is cloudy".

1. Atomic propositions, "It is raining" is $A$;
2. Negation of atomic propositions, "It is not raining" is $\neg A$;
3. Conjunction of propositions, "It is raining and it is cloudy" is $A \wedge B$;
4. Disjunction of propositions, "It is raining or it is cloudy" is $A \vee B$;
5. Implication of propositions and its decomposition, "If it is raining then it is cloudy" is $A \implies B$;
6. Removal of double negation, $\neg\neg A$ is $A$;
7. Distributive property of the *OR* connective, $(A \wedge B) \vee C$ is $(A \vee C) \wedge (B \vee C)$;
8. Negation of a conjunction, $\neg(A \vee B)$ is $\neg A \wedge \neg B$;
9. Negation of a disjunction, $\neg(A \wedge B)$ is $\neg A \vee \neg B$;
10. Application of the resolution rule, $Res((A \vee C), \neg A) = (P_1 - \{A\}) \vee (P_2 - \{\neg A\}) = C$;
11. Proving a theorem using the resolution rule with proof by contradiction.

KC2 KC3 and KC4 depend on learning KC1 and so on. Green KCs represent basic knowledge, yellow are novice skils, orange are intermediate and red is expert. When the user starts using the system, the ZPD is initialized with KC2, KC3 and KC4. In practice, exercises testing KC1 are impractical, as there

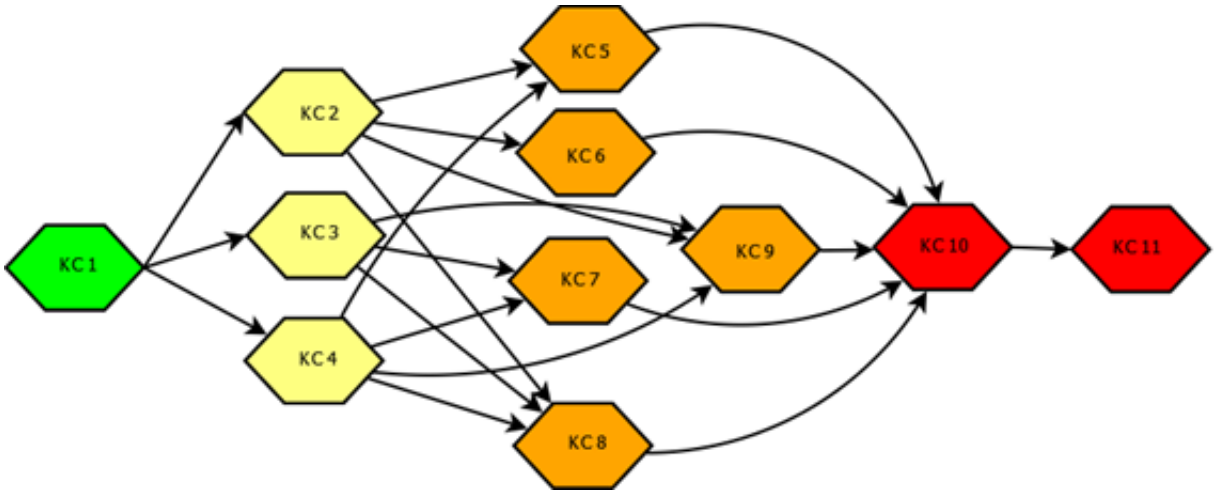


**Figure 3.3:** KC tree with skill dependency represented by the directional arrows

is only one possible exercise which is an atomic proposition, $A$ or any other letter, and, as other KCs depend on knowing KC1, we assume the user already knows it to start the system with the novice KCs unlocked. Both the division of the knowledge in different KCs and the KC tree were determined in the course of this work.

With this KC organization, the system goes through four phases of teaching:

1. Teaching novice skills of the basic logic connectives and proposition structure, ZPD includes KC2, KC3 and KC4;
2. Teaching intermediate skills of the IMPLIES connective and converting a proposition to its clausal form, ZPD includes KC5, KC6, KC7, KC8 and KC9;
3. Teaching application of the resolution rule to a sentence in the clausal form, by joining different clauses to achieve the empty clause, ZPD includes KC10;
4. Teaching a full resolution exercise, where a set of premises and a conclusion are given and checking whether the conclusion is proven by the premises, ZPD includes KC11.

### 3.2.2 Student model Module

For the student model, we define it as a list of 10 values, for every KC except KC1, corresponding to each KCs learning progress. This value is percentage based, ranging from 0%, where the KC has not been attempted yet, to 100%, where the KC has been fully taught. The ZPD is also included in the student model, initialized with KC2, KC3 and KC4, and, as KCs are learned by the user, it is updated with new KCs according to the KC tree in the figure 3.3.

After every activity, both the KC values and the ZPD are updated. Depending on the correctness $c$

of the exercise, which can be -1 for incorrect answers and 1 for correct ones, and on the learning speed $ls$, the expression [10] to update every KC value $u_{k,n}$ is the following:

$$u_{k,n+1} = u_{k,n} + ls \times c_{n+1} \tag{3.1}$$

With this approach, more importance is given to more recent inputs, as these are considered to better represent the student's knowledge. We can also adjust the learning speed to speed up or slow down the learning process. To give even more importance to recent inputs, we build on this method by including a parameter representing the streak $\sigma$ of exercises the student has gotten correct in a row, to a maximum of 10 exercises. Thus, the method becomes:

$$u_{k,n+1} = u_{k,n} + ls \times c_{n+1} + \alpha\sigma \tag{3.2}$$

As with the learning speed, the $\alpha$ parameter can be adjusted to make the streak more relevant or not. In practical uses, we use a small value for $\alpha$, for example 0.1 or 0.2, so at a maximum streak of 10 exercises in a row, it would not add more than 1 or 2 to the KC value. This parameter is introduced in order to show more difficult exercises to students who get many exercises right in a row, to provide a more challenging experience to more successful students.

## 3.3 Expert Knowledge Module

This module is built using *logic.py* library, with a change to the CNF conversion function. After all steps of CNF conversion, if we are left with an expression with repeated symbols in conjunction or disjunction with each other, for example $A \vee A$ or $\neg B \wedge \neg B$, these symbols are merged together, to $A$ and $\neg B$ respectively, as having both would be redundant. This module also includes a database of exercises testing every KC, developed during the course of this work. Depending on their content, these are classified according to which KCs they affect prioritizing the higher rated KCs. For example, for a proposition to be classified with KC2 it must only have a NOT connective, $\neg$, and for KC6 it must have two in succession, $\neg\neg$. As such, if an exercise is classified with KC6 we do not classify it with KC2. An example can be found in the exercise:

$$(\neg 1 \wedge \neg\neg 2) \wedge (\neg 2 \vee 1); KC3, KC4, KC6$$

Instead of letters, we use sequential numbers for each different letter and, if the exercise is shown to the user, all numbers are replaced with a random letter, for example $1 \implies 2$ can become $A \implies B$ or $Q \implies D$ or any other combination. This is done so the same exercise can be shown multiple times without looking repetitive. This exercise format is used for all activities except Full Resolution exercises,

where a distinction between the premises and the theorem we wish to prove is needed. For this exercise format, we separate both by a $\#$. If there are no premises or conclusion, we still present the exercise with a $\#$ but with nothing on the left side or right side respectively. For example, all the following propositions are valid exercises for Full Resolution:

$$A \vee B \wedge C \# C$$

$$\# C \implies (B \implies C)$$

$$(A \vee B) \wedge \neg(\neg A \implies B) \#$$

The exercises available in the system are mostly randomly generated, created using a script that randomizes the number of clauses and the number and order of logical connectives, keeping into account the validity of the generated expression. We chose this approach since there is a limited number of expressions that can be created using 2 or 3 different symbols with any combination of the four logical connectives. We provide a database with most, if not all, possible formulas with 2 or 3 symbols, and many exercises with 4 symbols. We also include some pre-selected exercises for KC10 and KC11 because, since it is necessary for most of the activities to be solvable, we include some exercises which have different steps when applying the resolution rule and are solvable.

## 3.4 Tutoring Module

This module is responsible for exercise selection and uses the ZPDES algorithm for such. With the KC values and the ZPD from the student model, we verify the ZPD for which KCs we can choose, then select a KC with the lowest value or one tied for the lowest value. With the KC chosen, we filter the exercises for the only exercises that have an impact on that KC and order them by length since, if an exercise has more clauses or symbols, it is considered harder. With the learning value for the chosen KC, taken from the student model, as $L_{kc}$ the number of exercises that impact that KC as $N_{kc}$ and the learning value at which we consider that KC to be taught as $Lmax_{kc}$, we choose an exercise based on the following expression:

$$ExerciseNumber = \frac{N_{kc} \times L_{kc}}{Lmax_{kc}} \tag{3.3}$$

With the exercise chosen, the activity must also be chosen, which depend on the KC and its learning value:

- Truth and False exercises are always picked for every novice and intermediate KC when they are below 25% taught and have a chance to be chosen between 25% and 75%, after this threshold, these activities stop being chosen;
- CNF conversion exercises have a chance to be chosen for every novice and intermediate KC when

they are over 25% taught and are always chosen when they are over 75%;

- Connecting Clauses exercises are only picked for KC10;
- Full Resolution exercises are only picked for KC11.

We use this selection of activities since we intend for the Truth and False exercises to be more prominent in the earlier stages of learning for students to gain familiarity to the expected output of converting a sentence to CNF. After this start, these exercises are replaced by CNF conversion, as it tests the same skills as Truth and False, but gives the student more agency and room for error. For the other two activities, we only select them for KC10 and KC11 because Connecting Clauses and Full Resolution, respectively, are pure applications of the knowledge in those KCs.

### 3.4.1 Error Diagnosis

For error diagnosis, we first check the validity of the answer given, by verifying there are no syntax issues or illegal characters or if there is an empty answer. After this verification there are two approaches we use for error diagnosis, one where we only check for correctness and an approach similar to *AITS* [5] diagnosis process, where we check for the edit distance and the type of error to give better feedback and update the KC values.

When measuring correctness, we compare the expression inputted, or shown in the Truth and False exercises, with the expected expression. If they include the same symbols and connectives in a similar order as the expected expression, for example $A \wedge B$ is equal to $B \wedge A$, the response is deemed optimal and correct. Otherwise, if there is a significant difference between the two formulas, the answer is deemed incorrect.

In the other approach, we verify the edit distance between the input and the ideal response. If there is any difference the answer is checked for its error. Here we make a distinction between basic errors, that students might make in the initial stages, and normal errors, where the student misses some steps of CNF conversion. Basic errors provide detailed feedback according to the type of error while normal errors provide feedback but also influence the KCs they reference by decreasing their learning value. The types of error are as following:

- Basic Errors:
  - Not enough clauses;
  - Too many clauses;
  - Clause not in original expression;
- Normal errors:

- KC5 error, if $\Longrightarrow$ is not simplified;
- KC6 error, if $\neg\neg$ two negation symbols are not cancelled out;
- KC7 error, if $\vee$ is not distributed to other conjunctions;
- KC8 error, if negation of a disjunction is not converted;
- KC9 error, if negation of a conjunction is not converted.

Firstly we verify the number of clauses to verify basic errors, then we check for any differing combination of symbols not present in CNF. For any error that is found, we display feedback on it and what steps the student missed. The full error diagnosis process can be found in the following figure 3.4.

With these two different error diagnosis processes, we have two distinct approaches when performing the KC value update and, by definition, the ZPDES algorithm. One approach where we only check for correctness and we simply use the ZPDES algorithm, and one where we use a more complicated error diagnostic process integrated with the ZPDES algorithm, where the errors of the user also affect their learning, since in this approach, we show exercises of the same type of those where the users make more errors.

## 3.5 Other Features

Some additional quality of life changes were also added to provide a better environment for students to perform activities. A help popup was implemented, which gives important information regarding the notation used for the system and other additional hints, found in the figure A.6. Also, a summary popup was implemented with information regarding the current streak of correct exercises and the solution to the last attempted exercise, found in the figure A.5.

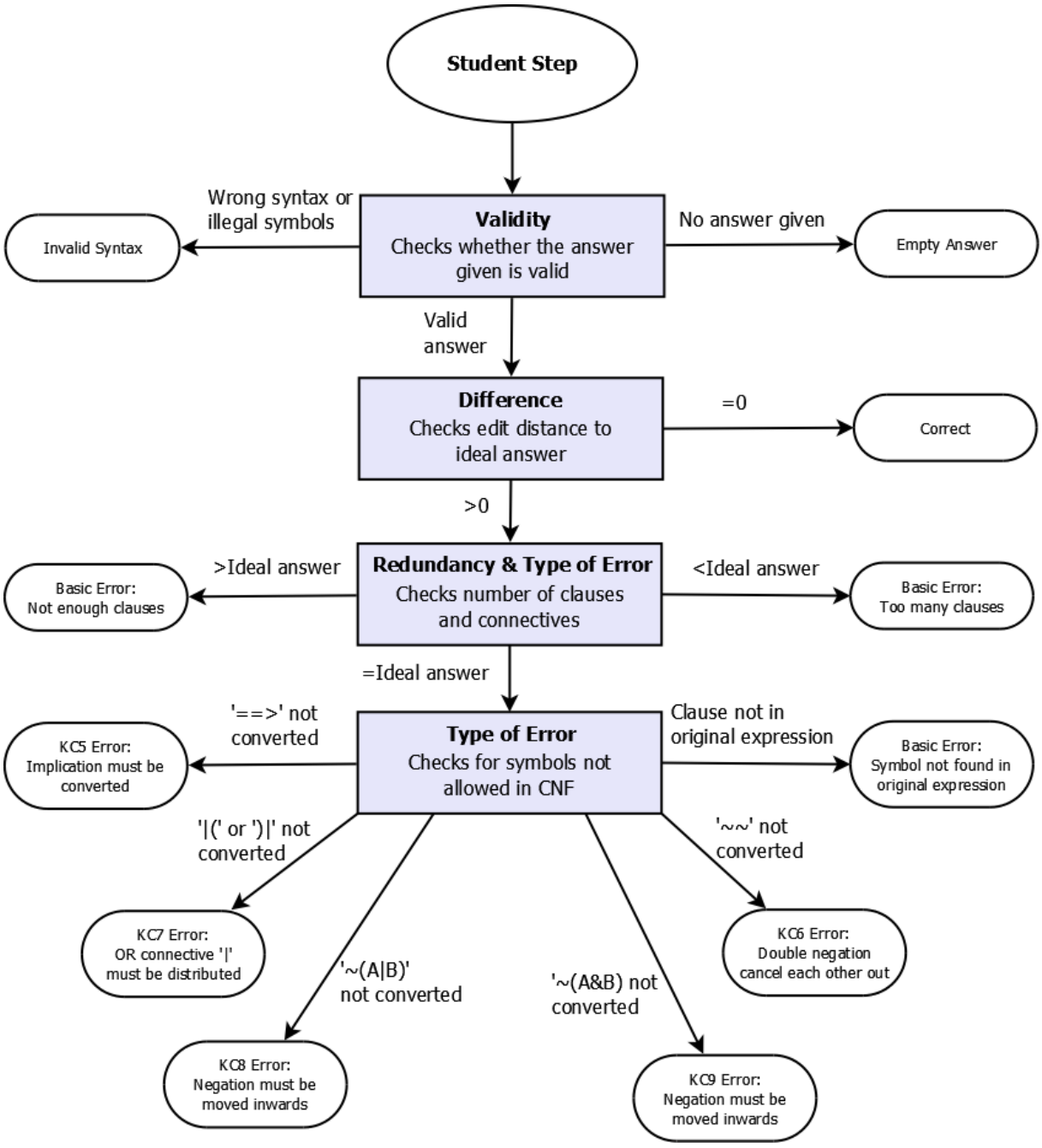


**Figure 3.4:** Full process of error diagnosis of P-res tutor

# 4

# User Study and Results

## Contents

## 4.1 Study Overview

In order to evaluate P-res tutor for its learning capabilities, more specifically, how much learning progress can be attained with it and how efficient it is at teaching, a user study was conducted. This study consisted of letting users interact with the system for a specified time and evaluating whether they had learned anything after, whether they were engaged during this interaction and whether the system is clear enough for users to understand without outside help.

Two different versions of the system were tested:

- P-res Tutor only with ZPDES algorithm;
- P-res Tutor with error diagnosis integrated with ZPDES algorithm.

Since it has already been proven that a system using the ZPDES algorithm has great learning potential in other systems [10, 23], we use this approach as our baseline, which we compare with the new approach using error diagnosis. With this comparison, we are able to analyse the impact of error diagnosis in the user's teaching, as we expect students using error diagnosis to achieve better results due to better feedback when performing the system's activities.

### 4.1.1 Description

The study was conducted using fourteen participants, seven for each version, with most being current or former students of Engineering in Instituto Superior Técnico. Instead of only selecting students of Computer Engineering, we included students from various different courses, such as Mechanical Engineering, Civil Engineering and Electrical Engineering, with most being second year students. In this way, we also manage to assess whether the system can teach a random college-level student a subject normally reserved for computer science students.

Before the study, every participant was given a short theoretical introduction to Propositional Logic, including the meaning of propositions, logical connectives, how to convert a sentence to CNF and how to apply the resolution rule to two propositions. For non-computer science students, this is the first contact they have with logic, more specifically, propositional logic (although some participants commented that they still remembered some logic from learning Philosophy in high school). Thus, we use the system to consolidate the knowledge we introduce in a short, roughly fifteen minutes, lesson.

After this introduction, we also present the interface and the activities of the system to reduce errors unrelated to learning logic. We then give unrestricted access to the system during a minimum of thirty minutes for each user, one at a time, starting with an initial student model and the stream of exercises. While the student hadn't learned everything, the stream kept showing new exercises. After the minimum time had passed, the student had the choice to stop the study or keep going to finish their learning.

Assistance was also provided for the student for any question regarding use of the system while the test was being carried out.

At the end of the study, either from request of the student or from finishing the system, general questions about the interface, the exercise selection and how much propositional logic they have actually learned were asked and the answers recorded. Pen and paper were also supplied to each user.

## 4.2 Results

To evaluate each student's learning progress, our first approach was to register how many KCs had been learned after the interaction but, since most of the participants wished to finish the system and achieve all KCs by continuing to use the system after the thirty minutes, we instead chose to measure the percentage of correctness of each student for both approaches and compare the two populations of students 4.1. This decision was also made because it was important for students to interact with all activities and two of them, Connecting Clauses and Full Resolution Exercises, are only found in the later stages of the system's learning, making it very difficult for a student to reach that stage with a time limit of thirty minutes. Students who wished to complete the system finished the course with a time of interaction of forty-five minutes to one hour. For students who wished to end the study after the minimum time, which were only students 5 and 11, the last ten minutes of the study were dedicated to KC10 and KC11 exercises only. We also measured the overall correctness of each KC to evaluate whether proper feedback and error diagnosis influence certain KCs learning 4.2.

Between both approaches, the better average results were found in the ZPDES plus Error Diagnosis system, with an average correctness percentage of 92%. In comparison, the pure ZPDES approach had a smaller average correctness of 83%. The better results in the pure ZPDES approach, students 1, 2 and 4, were all current Computer Engineering students, meaning they could accurately identify their errors on their own. In contrast, the worse results, with the worst being a correctness rate of 74%, were students that had just learned logical expressions, CNF conversion and the resolution rule for the first time in a short introduction and, thus, made several consecutive errors. In the ZPDES plus Error Diagnosis system, the same difference between students happened, where the best results came from current or former Computer Engineering students, although the difference between the higher and lower results was much lower. Results for this approach were generally higher, as even the lowest correctness, 86%, was higher than the average correctness for the pure ZPDES approach.

Regarding KC correctness, the lower KCs for the pure ZPDES approach were KC5 with 79%, KC7 with 63%, KC8 with 82% and KC9 with 71%. As these KCs teach conversion to the clausal form, without proper feedback it was expected for these KCs to require numerous tries for students to understand where they went wrong. In comparison, the KC correctness for the ZPDES with Error Diagnosis was

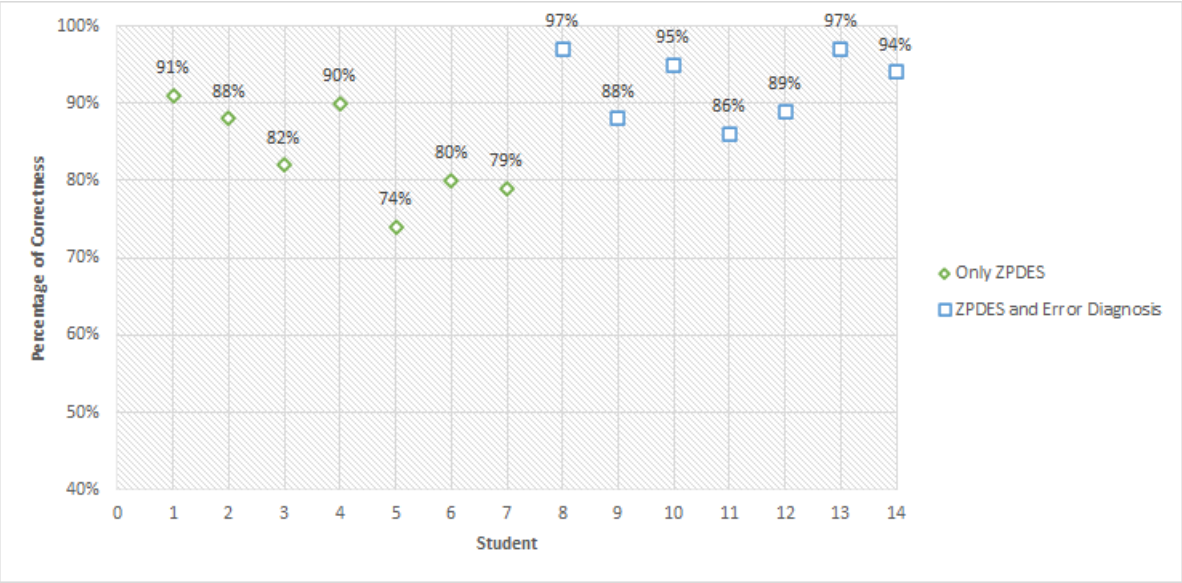


**Figure 4.1:** Graph showing the results of the correctness for each student in the user study

much better, as some of the previous lowest KCs were now with higher values in correctness, for example KC7 with 83% and KC9 with 95% being the best improvements. For KC11, which is the last KC to be learned, we also saw a great improvement, from 86% to 96%. All of these results show that there is an influence of the error diagnosis process in the learning of the system and the learning of certain KCs.

## 4.3 Discussion

By only looking at the statistical results, we can clearly see the ZPDES plus Error Diagnosis system provided better results and was a more efficient approach since the students using this version achieved the same learning progress with less exercises. These results were expected as this approach provided a better environment for students to not repeat their errors, as proper feedback allowed students to understand where they had made their mistake and correct it in the future. This is also proven by the KC correctness values, as KCs that were harder to understand without feedback were now easier to attain and had a better correctness value.

In general, students seemed engaged when using the system, proven by the fact that most wished to end the training even when told the minimum time had passed. As such, these same students commented that they considered they had actually learned the basics of propositional logic and how to perform Propositional Resolution.

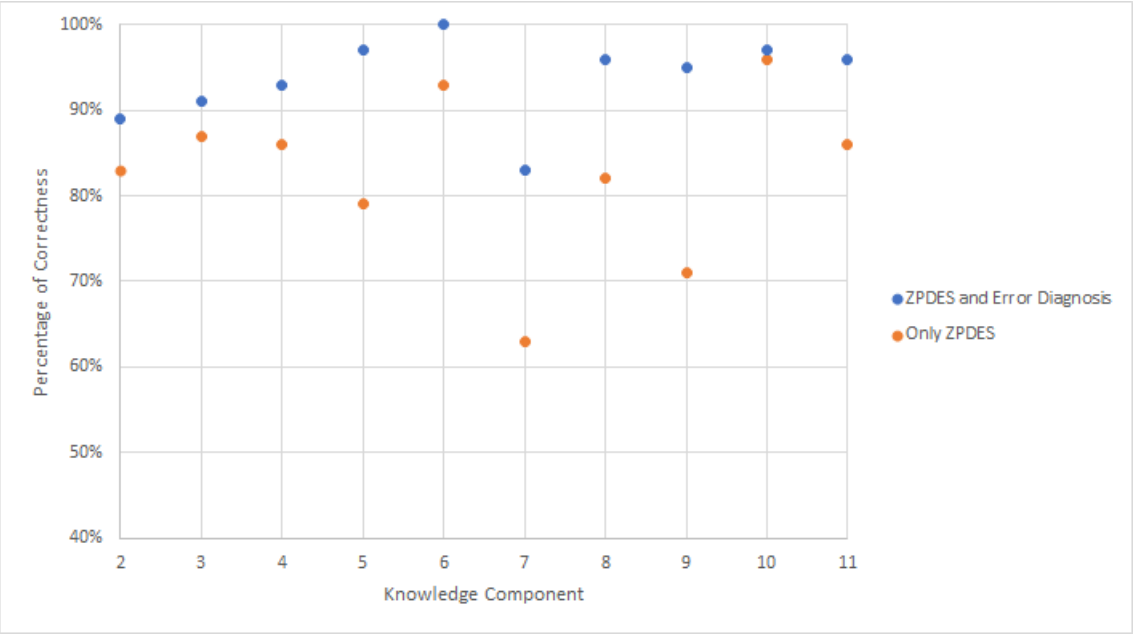


**Figure 4.2:** Graph showing the average correctness of each KC for each approach

### 4.3.1 General Comments

Comments were generally positive, although some criticisms of the system were also noted. Students who used the Error Diagnosis version of the system commented that the feedback and explanation of the previous exercise's solution was clear and concise. Users also complimented the clean interface and some commented that they considered the exercises were tailored for their needs.

Regarding criticisms, one of the most prominent opinions was that the system was not very user friendly, as activities and the descriptions of what to do were a bit vague, which left them unsure of what to do. This difficulty was pointed out mostly about the Connecting Clauses activity since it resembled a multiple choice activity, which required outside assistance for some students to understand how to complete. An idea by one of the users was to include a video tutorial of each activities execution in a separate tutorial function, which would, in theory, clarify how to interpret the exercises description.

Another criticism was related to a lack of a sense of progress to the stream of exercises. A common comment during user testing was if the system would endlessly present exercises or if it had an end, since the only feedback received was related to the user's input and no feedback is given related to how much learning progress the student has achieved. A possible solution, also suggested by a student, is to include a counter of exercises in the main page of the exercise, including how many are left for each KC to be completed and, by definition, for the course to be completed.

Overall, this study was conclusive in evaluating our both approaches with college students, although a more extensive study, with a more diverse population of different backgrounds and ages would provide

more conclusions about the system, its advantages and its shortcomings.

# 5

# Conclusion

## Contents

## 5.1 Achievements

Concluding this dissertation, we sum up the achievements made with this work. Our goals included building a functioning ITS that teaches Propositional Logic and Propositional Resolution by adapting to each student's different capabilities, by providing a personalized learning experience without any previous knowledge or assumptions regarding the student.

For these goals, other sub-objectives were also required. We defined each individual skill, as knowledge components, needed to learn Propositional Resolution and defined the hierarchy between them. We also defined the student model as a combination of learning values associated with each KC and the ZPD, which serves as the sub-set of skills the student has more learning potential.

Using the python library *logic.py*, we built a framework for propositional logic exercises on a Flask application and generated a near-endless supply of logic exercises to include in the system. For the selection of these exercises, we used the ZPDES algorithm and we also managed to improve the algorithm's exercise selection by integrating it with a more complicated error diagnosis process. This process was also created in the context of the dissertation, by verifying the difference between the student answer and the ideal answer and checking for incorrect combinations of symbols in the student answer.

Finally, we conducted a user study to evaluate the system's teaching potential with students at college-level. This study proved that the system is capable of this, as most students who participated were capable of performing Propositional Resolution after the interaction. The study also showed our approach using ZPDES with our Error Diagnosis process to be more efficient of the pure ZPDES approach, as the correctness results from the students using the ZPDES plus Error Diagnosis version were better, meaning they achieved the same learning progress with less exercises.

In summary, we managed to create P-res Tutor, an ITS that is able to provide a customizable and adaptive experience for any student to learn Propositional Resolution.

## 5.2 Future Work

Regarding future work in this area, most of it is regarding improvements in the existing system or extensions of P-res tutor. Further work can also be done researching other ways of exercise selection, for example using the RiaRit algorithm.

### 5.2.1 Improvements

In terms of improvements for P-res Tutor, the priority would be to improve on the existing activities to make them more user friendly, by changing the activities descriptions or even the activities themselves. A possible improvement would be to change the form of input in CNF conversion exercises, instead of

being text-based, all of the possible symbols would be in selection boxes and the input for the exercise would be created by clicking these boxes. The only symbols that would appear in these boxes would be the logical connectives and the propositions in the original sentence. This would remove doubts over what is possible in CNF conversion and what the expected output is at the cost of taking some freedom from the system.

The error diagnosis process could also be improved, as it now only includes error detection for five KCs. This process should extend to all KCs and to all activities as we are not currently able to diagnose the error in the Connecting Clauses exercise. A more detailed process would find an ideal sequence of connecting clauses to achieve the empty clause and compare this sequence to the user inputted one, while checking for any redundant steps.

When it comes to the adjustable parameters used in the exercise selection function, we could also improve on these by finding the ideal values for them. This would require further user studies so we could test the impact of each parameter, for example the impact of the learning speed, of the streak of exercises or of the error diagnosis. A simulation using virtual students would also allow us to better evaluate the system.

Finally, as said before, the exercise selection could be improved by implementing the RiaRit algorithm with our developed Error Diagnosis process, and compare it with the ZPDES approach, to evaluate whether it would be more efficient than the existing solution. This algorithm would also require a more detailed student model, including competence values for each activity.

### 5.2.2 Extensions

The priority when thinking of extensions for P-res tutor is including new activities in the system. One possible activity would be a truth table exercise. By presenting a truth table with empty spaces, the student would have to fill out the rest of the table with the correct values. This would help us better teach the basic KCs, such as the knowledge of what every logical connective means in a proposition. This would also allow the system to teach all of Propositional Logic and not only Propositional Resolution.

A major extension would be to implement First-Order Logic and First-Order Resolution in the system. This would require some extensions to the $logic.py$ library, as it only includes a framework for Propositional Logic. It would also require implementation of the existential quantifier $\exists$ and the universal quantifier $\forall$ and their decomposition to CNF. Furthermore, both different KCs and a different KC tree would have to be created since First-Order Logic includes many concepts, such as predicates, which Propositional Logic does not consider. For this extension, help could be obtained from analysing ITSs which test First-Order Logic [12].

# Bibliography


[1] T. Murray, “Authoring Intelligent Tutoring Systems : An Analysis of the State of the Art,” 2007.

[2] R. Nkambou, J. Bourdeau, and V. Psyché, “Building intelligent tutoring systems: An overview,” *Stud. Comput. Intell.*, vol. 308, pp. 361–375, 2010.

[3] B. Clement, D. Roy, P.-Y. Oudeyer, and M. Lopes, “Multi-Armed Bandits for Intelligent Tutoring Systems,” vol. 7, no. 2, pp. 20–48, 2013. [Online]. Available: http://arxiv.org/abs/1310.3174

[4] S. Russell, S. Russell, and P. Norvig, *Artificial Intelligence: A Modern Approach*, ser. Pearson series in artificial intelligence. Pearson, 2020. [Online]. Available: https://books.google.com.br/books?id=koFptAEACAAJ

[5] F. Grivokostopoulou, I. Perikos, and I. Hatzilygeroudis, “An Educational System for Learning Search Algorithms and Automatically Assessing Student Performance,” *Int. J. Artif. Intell. Educ.*, vol. 27, no. 1, pp. 207–240, 2017. [Online]. Available: http://dx.doi.org/10.1007/s40593-016-0116-x

[6] W. J. Clancey, “Intelligent Tutoring Systems: A Survey,” *Explor. Artif. Intell.*, pp. 1–43, 1986.

[7] H. S. Nwana, “Intelligent tutoring systems: an overview,” *Artif. Intell. Rev.*, vol. 4, no. 4, pp. 251–277, 1990.

[8] D. Abraham, L. Crawford, L. Lesta, A. Merceron, and K. Yacef, “The logic tutor: A multimedia presentation,” vol. 3, 01 2001.

[9] S. Lukins, A. Levicki, and J. Burg, “A tutorial program for propositional logic with human/computer interactive learning,” *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.*, pp. 381–385, 2002.

[10] M. S. Reis, “RegexTutor : A Fully Online Intelligent Tutoring System,” September, 2020.

[11] F. Grivokostopoulou, I. Perikos, I. Hatzilygeroudis, F. Grivokostopoulou, I. Perikos, I. Hatzilygeroudis, A. Tools, F. Grivokostopoulou, I. Perikos, and I. Hatzilygeroudis, “Assistant Tools for Teaching

FOL to CF Conversion To cite this version : HAL Id : hal-01521414 Assistant tools for teaching FOL to CF Conversion," 2017.

[12] F. Grivokostopoulou, I. Perikos, and I. Hatzilygeroudis, "An intelligent tutoring system for teaching FOL equivalence," *CEUR Workshop Proc.*, vol. 1009, pp. 20–29, 2013.

[13] C. Galafassi, F. Galafassi, E. Reategui, and R. Vicari, "EvoLogic: Sistema Tutor Inteligente para Ensino de Lógica," pp. 222–233, 2020.

[14] H. Mandl and A. Lesgold, *Learning Issues for Intelligent Tutoring Systems*, 1st ed., 1989.

[15] J. Crawford and U. of Sydney., *EMYCIN : an expert system shell / James Crawford*. Basser Dept. of Computer Science, University of Sydney [Sydney], 1987.

[16] S. Stankov, M. Rosić, B. Žitko, and A. Grubišić, "Tex-sys model for building intelligent tutoring systems," *Computers & Education*, vol. 51, no. 3, pp. 1017–1036, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131507001297

[17] N. T. Heffernan, T. E. Turner, A. L. N. Lourenco, M. A. Macasek, G. Nuzzo-Jones, and K. Koedinger, "The assistment builder: Towards an analysis of cost effectiveness of its creation," in *FLAIRS Conference*, 2006.

[18] A. Munro, *Authoring Simulation-Centered Learning Environments with Rides and Vivids*, 01 2003, pp. 61–91.

[19] S. Ainsworth, N. Major, S. Grimshaw, M. Hayes, J. Underwood, B. Williams, and D. Wood, *REDEEM: Simple Intelligent Tutoring Systems from Usable Tools*. Dordrecht: Springer Netherlands, 2003, pp. 205–232. [Online]. Available: https://doi.org/10.1007/978-94-017-0819-7_8

[20] R. V. Lindsey, J. D. Shroyer, H. Pashler, and M. C. Mozer, "Improving students' long-term knowledge retention through personalized review," *Psychological science*, vol. 25, no. 3, pp. 639–647, 2014.

[21] K. Shabani, M. Khatib, and S. Ebadi, "Vygotsky's zone of proximal development: Instructional implications and teachers' professional development." *English language teaching*, vol. 3, no. 4, pp. 237–248, 2010.

[22] A. Slivkins, "Introduction to multi-armed bandits," *Found. Trends Mach. Learn.*, vol. 12, no. 1-2, pp. 1–286, 2019.

[23] B. Clement, P. Y. Oudeyer, and M. Lopes, "A comparison of automatic teaching strategies for heterogeneous student populations," *Proc. 9th Int. Conf. Educ. Data Mining, EDM 2016*, pp. 330–335, 2016.

[24] A. T. Corbett and J. R. Anderson, “Knowledge tracing: Modeling the acquisition of procedural knowledge,” *User modeling and user-adapted interaction*, vol. 4, no. 4, pp. 253–278, 1994.

[25] K. VanLehn, “The Behavior of tutoring systems,” *Int. J. Artif. Intell. Educ.*, vol. 16, no. 3, pp. 227–265, 2006.

[26] R. van der Bent, J. Jeuring, and B. Heeren, “The diagnosing behaviour of intelligent tutoring systems,” in *European Conference on Technology Enhanced Learning*. Springer, 2019, pp. 112–126.

[27] F. F. P. Galafassi, J. C. Gluz, L. Gomes, and M. Mossmann, “Sistema Heráclito: Objeto de Aprendizagem para o Ensino da Dedução Natural na Lógica Proposicional,” no. Cbie, pp. 249–258, 2013.

# A

# System Screenshots

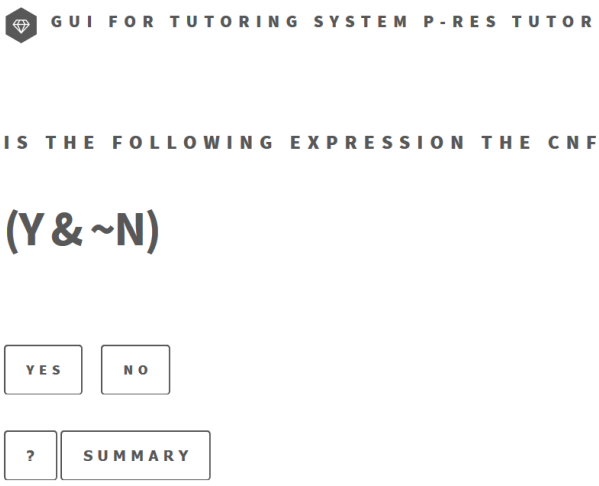


**Figure A.1:** Truth and False exercise

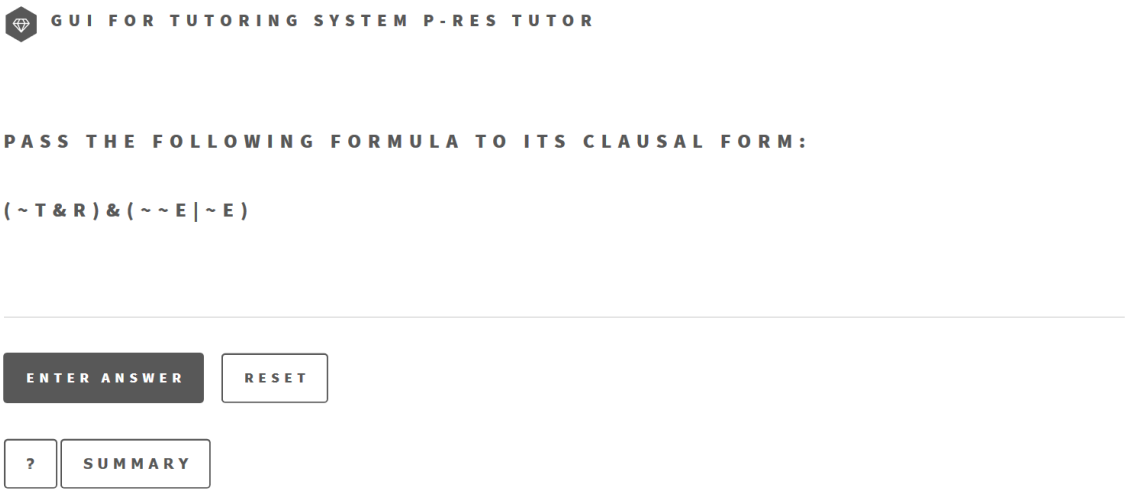


**Figure A.2:** CNF conversion exercise

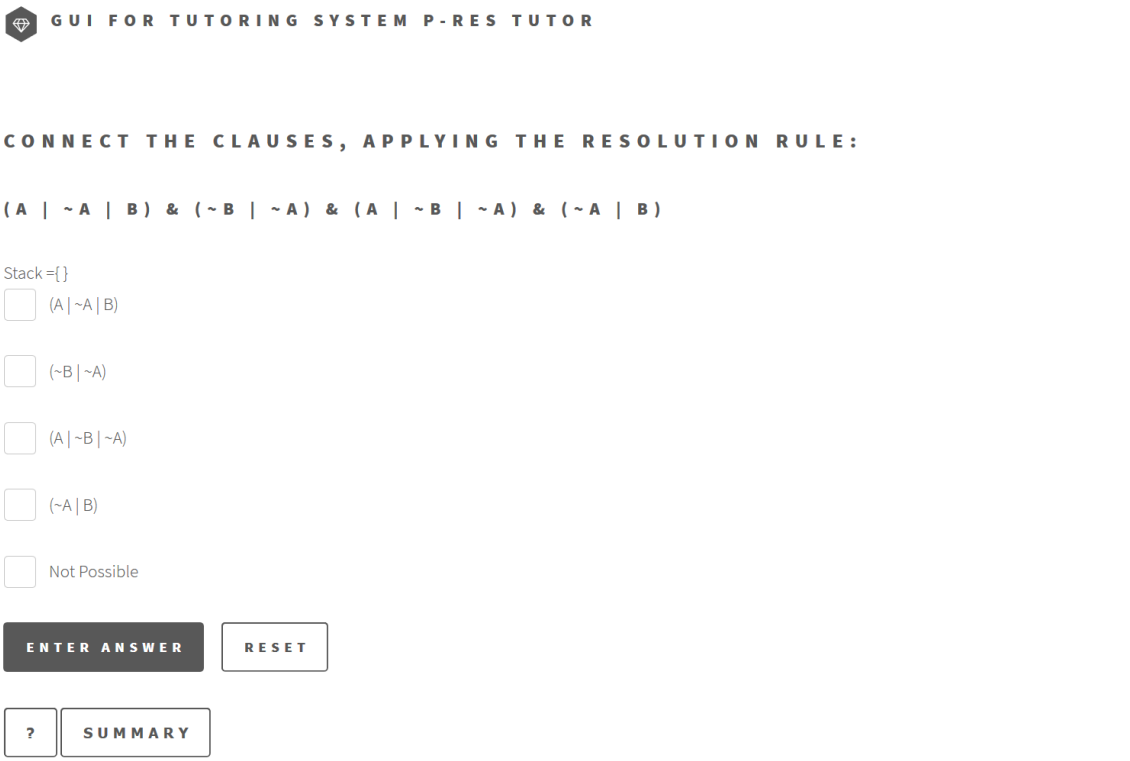


**Figure A.3:** Connecting Clauses exercise

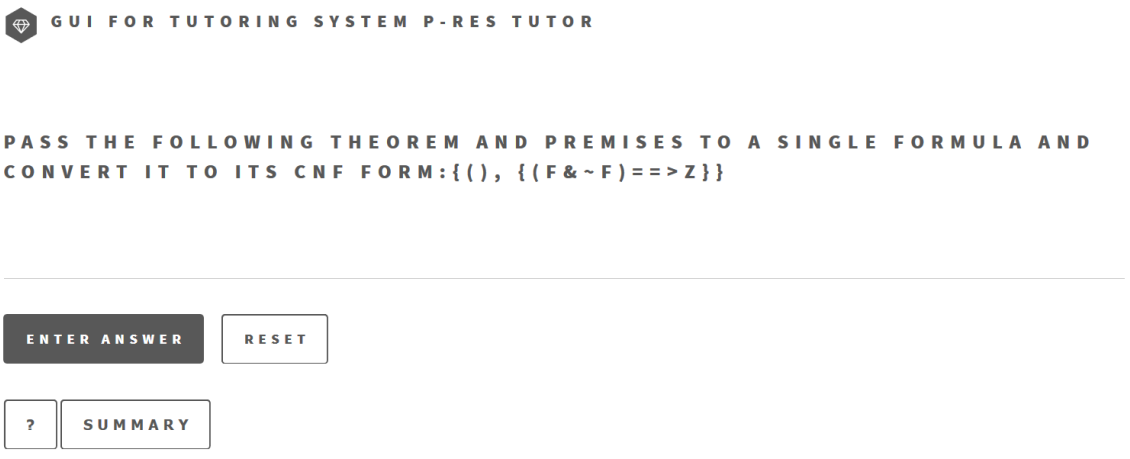


**Figure A.4:** Full Resolution exercise

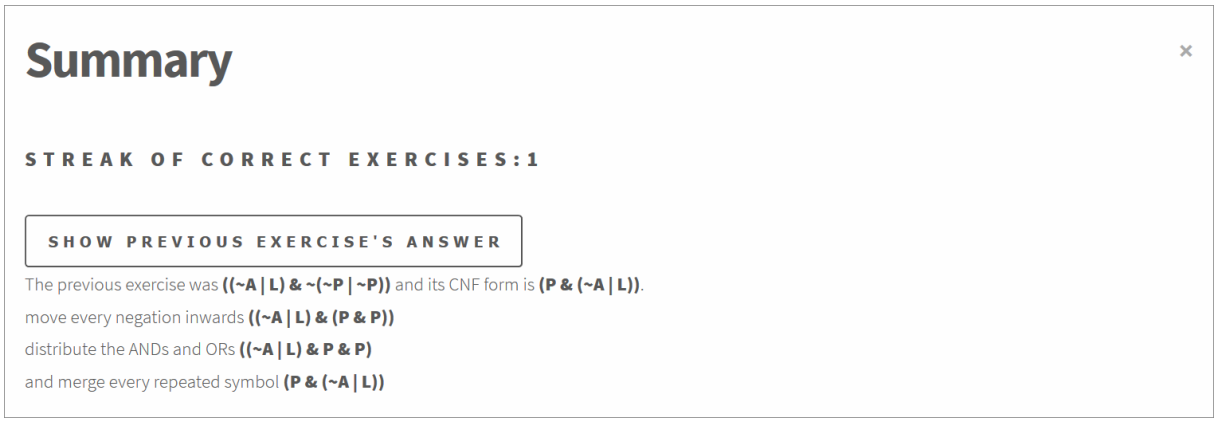


**Figure A.5:** Summary popup, with streak of exercises and correction of last exercise

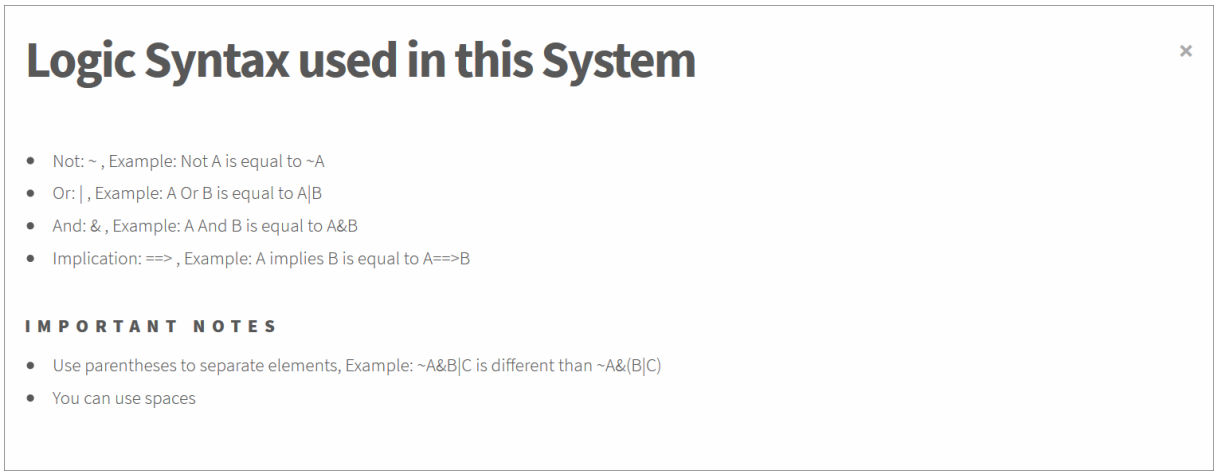


**Figure A.6:** Help popup, with notation used by the system and other useful tips