Journal of Universal Computer Science, vol. 14, no. 20 (2008), 3282-3306
submitted: 2/7/08, accepted: 16/11/08, appeared: 28/11/08 © J.UCS

A Tool for Reasoning about Qualitative Temporal Information: the Theory of S-languages with a Lisp Implementation

Irène A. Durand
(LaBRI, CNRS, Université de Bordeaux, Talence, France
idurand@labri.fr)

Sylviane R. Schwer
(LIPN, CNRS, Université Paris-Nord, Villetaneuse, France
schwer@lipn.univ-paris13.fr)

Abstract: Reasoning about incomplete qualitative temporal information is an essential topic in many artificial intelligence and natural language processing applications. In the domain of natural language processing for instance, the temporal analysis of a text yields a set of temporal relations between events in a given linguistic theory. The problem is first to express events and any possible temporal relations between them, then to express the qualitative temporal constraints (as subsets of the set of all possible temporal relations) and compute (or count) all possible temporal relations that can be deduced. For this purpose, we propose to use the formalism of S-languages, based on the mathematical notion of S-arrangements with repetitions [Schwer, 2002]. In this paper, we present this formalism in detail and our implementation of it. We explain why Lisp is adequate to implement this theory. Next we describe a Common Lisp system `SLS` (for S-LanguageS) which implements part of this formalism. A graphical interface written using McCLIM, the free implementation of the CLIM specification, frees the potential user of any Lisp knowledge. Fully developed examples illustrate both the theory and the implementation.

Key Words: Temporal reasoning, relation algebras, S-languages, Lisp
Category: I.2.4, I.1.1, F.4

1 Introduction

The notion of time is ubiquitous in any activity requiring intelligence. In particular, several important notions like change, causality, and action are described in terms of time. Time has been recognized as a fundamental notion in modeling and reasoning about changing domains. Reasoning about temporal constraints is thus an important task in many areas of computer science and elsewhere, including scheduling, natural language processing, planning, database theory, diagnosis, circuit design, archeology, genetics, and behavioral psychology [David et al., 2005].

Let us take two examples whose purpose is firstly to illustrate our subject throughout the paper and secondly to give an idea of what kind of problems can be solved within our framework.

1.1 The trains example

Consider a set of 6 trains named $\{A, B, C, D, E, F\}$ with the following set of temporal constraints[1].

1. A, B and E reach the platform at the same time
2. A leaves before B.
3. A leaves after or at the same time as C but before the arrival of D.
4. D and F arrive at the same time as B is leaving.
5. E and D leave at the same time.

Due to security reasons, a train is not allowed to arrive on a track from which another train is currently leaving; therefore, we consider the following problem: How many platforms are necessary to satisfy constraints 1 to 5?

1.2 Fred's example

This second example comes from natural language processing. In natural languages, temporal constraints are generally vague or/and underspecified or indefinite, as shown by the following example taken from [van Beek, 1990].

> Fred was **reading the paper** *while* **eating his breakfast**.
> He put the **paper down *and* drank the last of his coffee.**
> ***After* breakfast, he went for a walk.**

Van Beek's linguistics analysis provides the following specification:

– a set of four lasting items (intervals), corresponding to **eating breakfast**, **reading the paper**, **walking** and **drinking coffee**,

– knowledge about relationships between these intervals, in terms of sets of binary relations between the intervals.

The questions we may ask then are: Is there a model (a way of completely relating these items) which satisfies the constraints? Can we exhibit a possible model, all possible ones?

To answer these questions it is necessary to formalize them. Many frameworks for formalizing time have been proposed, all based on works by logician philosophers who were concerned with physics, psychology or language theory, among whom Frege, Prior, Montague, Hamblin, Reichenbach or Russell, Whitehead and Nicod [Hamblin, 1969, Reichenbach, 1947, Russell, 1914, Whitehead, 1920, Nicod, 1923]. This explains why most of these works have been handled inside a logical framework [Krokhin et al., 2003, David et al., 2005].

There are two main approaches for defining a temporal ontology. The first one is based on the absolute Newtonian Time point of view. Time is a primitive concept, given

[1] This example is inspired by [Revault, 1996].

a priori and flowing uniformly in measurable lapses. All phenomena have an extension in time, given in terms of instants or stretches of time, or both. The other approach is based on the relative Time point of view. In this approach, the Time concept derives from the primitive concept of event. This approach traces back to Leibnitz and is the one followed by the *natural* philosophers[2] Whitehead, Nicod, and Russell.

Works on temporal reasoning investigate both approaches, taking as temporal items either points, intervals or events from which points and/or intervals are built, or both points/intervals and events with functions which associate to each event a set of points and/or intervals. Time is described as an ordered structure, whose elements are points or intervals. This structure is endowed with properties which are related to linearity, bounds, and density. The choice of non linearity disconnects the description of the Time from the phenomenal world, but relates it to a collection of hypothetical worlds including the phenomenal one.

Temporal reasoning is an essential task that aims at finding a/some/all possible orderings between events or states given a set of constraints. Any domain with elements which can be ordered and for which we can describe ordering constraints can be handled in the same way, for instance, a set of actions with a causality relation.

A temporal constraint is said to be *qualitative* if it is only concerned with the relative temporal relations between items, and not with their respective durations, which is the quantitative aspect. In this article, we are concerned with the qualitative aspect of temporal reasoning. In this context, two problems arise:

1. a representation problem: how to represent time and temporal items, and which temporal relations are to be represented and how,
2. a calculus problem for reasoning: for example, knowing that A and B are in relation R_1 and that B and C are in relation R_2, which possible relations are derivable between A and C?

Temporal reasoning derives from the mathematical theory of the Boolean calculus of relations, which is included inside a first order theory of logic and was first introduced by de Morgan in 1864 and then developed by Peirce and Schröder at the very end of the 19th century. In 1941, Tarski [Tarski, 1941] reformulated this calculus as an algebraic theory called *relation algebra*[3] [Jonsson and Tarski, 1952]. Ladkin and Maddux [Ladkin and Maddux, 1994] introduced temporal reasoning into this specific framework.

In this framework, any relation is a binary one. Temporal relations can be derived from the two fundamental relations of *precedence* and *simultaneity*. Precedence is inherently a binary relation, but it is not the case for simultaneity, which can be defined

[2] These philosophers asserted that time is built from nature and that a moment is a ”passage of nature” [Whitehead, 1920], that is, the set of all events occurring simultaneously at that moment.

[3] This should not be confused with the algebraic formulation of operations on relational data bases, known as relational algebra, which is close to another theory also developed by Tarski under the terms cylindrical algebra. But the Join-operator, that we have introduced for S-languages is directly derived from the Join operation between databases relations.

with any number of arguments. The description of the precedence relation in terms of sequences of symbols (words) is very natural and will be presented in the following. It is already used in other frameworks such as path expressions [Campbell and Habermann, 1974], Petri nets [Peterson, 1983] and trace languages [Pratt, 1986]. However, plain words are unable to express simultaneity and in those frameworks only commutation and concurrency can be handled: doing A and B simultaneously is not the same as doing A and B independently and cannot be described with a sequence.

The raison d'être of the S-Languages[4] is to represent and calculate these two fundamental relations on arguments having a representation in terms of sequence of letters. The S-languages formalism addresses the two problems mentioned above: the representation and the calculus problems, from a completely new point of view. This framework is neither based on points/intervals nor on events representations, but on the notion of identity of temporal items. This allows to depict the temporal extension of temporal items and qualitative temporal relations in an homogeneous way, that is, with sequences of sets of symbols, viewed as an extension of the formal languages theory. Here, all temporal items are described in the same way, hence, there is a unique qualitative temporal calculus.

Let us now give a brief and informal description of S-languages. It was first presented in 1995[5]. A letter `a` is associated with each temporal item A (event, or fact or state) and is used as its identity. Any item can be characterized as event-like, lasting or repetitive. Let us denote this characteristic as the temporal *aspect* of the item. If a temporal item A is event-like, only one occurrence of its identity will be used; if A is lasting, *two* occurrences of its identity will be needed, each one representing one bound of its interval of duration. If A is lasting and repeated n times, it will be represented by $2n$ occurrences of its identity. In a sequence of letters (a word), the fact that a letter `b` is after another letter `a` expresses *precedence* between `a` and `b`. To express simultaneity, we define *S-letters* which are sets of letters occurring at the same time; an S-letter models a *specific* moment (where an item happens, starts, or ends). *S-words* are words over S-letters; an S-word models a sequence of specific moments. Assuming that B and C are both temporal lasting items, the S-word $w_1 = [\{b,c\}\{c\}\{b\}]$ contains two pieces of information: B and C start at the same time and B finishes after C. Assuming that A is an event-like temporal item, the S-word $w_2 = [\{b\}\{a\}\{b\}]$ means that A occurs during the lasting item B. A graphical representation of w_1 and w_2 is given in Figure 1. To simplify the presentation of this work, we shall restrict ourselves to handle **lasting** items – that is lasting and repetitive lasting objects – although taking into account event-like objects does not induce major difficulties.

[4] S for Set languages in general or for Synchronization or Simultaneity in the framework of time.

[5] During the first KANEOU seminar of the Research Group of CNRS for spatial and temporal reasoning, with the title "Dépendances temporelles: les mots pour le dire" but only published in [Schwer, 2002].

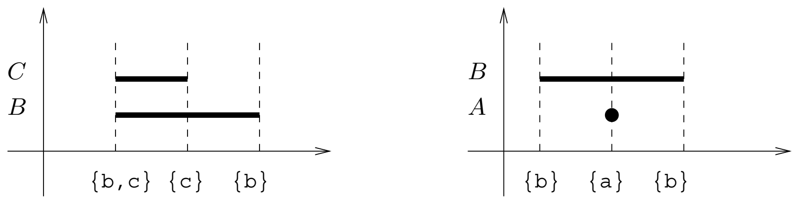


Figure 1: Representation of S-words w_1 and w_2

The S-words formalism has several advantages. It provides a very compact way of representing temporal information and a powerful tool for reasoning with this representation: the so-called *join* operation. A particular case, the mix operation (which represents the lack of constraints between items) offers in some cases a way of avoiding an exponential blow-up.

Section 2 present the two problems quoted in the introduction: the representation and the reasoning problems. Section 4 presents in detail the S-languages framework. Section 5 shows how temporal reasoning can be achieved using the S-languages formalism. Section 6 shows how the Lisp language is very well suited to implement the S-languages framework and presents the implementation aspects. The article ends with an overview on some related works and perspectives.

2 Representation and reasoning

2.1 Representation

This section deals with the representation problem; the first problem listed in the introduction.

2.1.1 Representation of relative relations

Temporal primitives are abstract elements whose purpose is to translate temporal items and their relationships according to the classical spatial representation of time, that is, on a geometrical oriented line. If the item is an instantaneous event, a *point* is used, if the item has a measurable duration, an *interval* is chosen as a temporal primitive. If the item is a sequence of occurrences of other items, a *chain* of points or/and intervals can be used; hence, relations are derived from four types of relations: (i) between two points, (ii) between two intervals, (iii) between a point and an interval, and (iv) between an interval and a point (the inverse of (iii)).

Inside the relation algebra, and therefore in the temporal reasoning area, a relation is in fact a collection of basic or atomic relations. For instance, 'before or simultaneous' is a temporal relation composed of the disjunction of two atomic relations 'before' and 'simultaneous', in the same manner as $\leq$ is read $<$ or $=$. When temporal relations are achieved inside the classical spatial representation of time, the relation algebra can be called a *situation calculus*, as defined by Leibniz. For the sake of clarity, we call an atomic situation, a (relative) situation. A relation is then a disjunction or a subset of the set of situations. The three possible situations between points are shown in Figure 2.

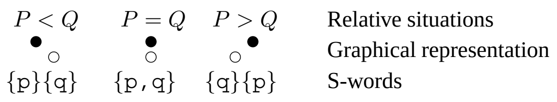


Figure 2: The three relative situations between two points on a line

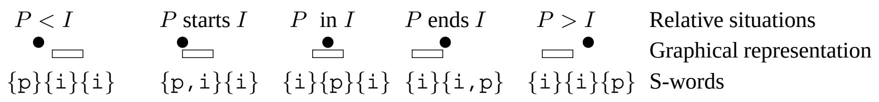


Figure 3: The five relative situations between a point P and an interval I on a line

Figure 3 shows the five possible situations between a point and an interval. Placing two intervals on the time line, regardless of their length, gives the thirteen relations which are shown in Figure 4, together with the names given by Allen [Allen, 1983], to denote these situations. The figures include the S-word representation which was briefly presented in the introduction and which will be presented in greater details in Section 4.

Situations between chains of intervals or points are derived from the situations between intervals; some frequent situations are named like the one depicted in Figure 5.

It is easy to see that situating relatively two or more temporal items in that framework is just the same as situating relatively chains of points. For instance, situating relatively n intervals is equivalent to situating relatively n chains of two points.

More generally, a relative situation between n temporal items $I_1 \cdots I_n$, each item I_k described by π_k points and ρ_k intervals is a relative situation between n chains of respectively $\pi_1 + 2\rho_1, \cdots, \pi_n + 2\rho_n$ points.

At this time, it becomes natural to associate an identity – a letter – to each chain, and to associate an occurrence of the letter identifying a chain to each of its points. Even more, the spelling of a word from left to right is in the natural correspondence with the relative positions of points on the time line, but the synchronized ones. It is the purpose of S-languages to overcome this synchronization problem. A natural question is to count the number of possible relative situations between n chains containing respectively $p_1, \ldots, p_n$ points. This number is given by the generalized Delannoy number $D(p_1, \ldots, p_n)$, given by the following recursive function [Schwer, 2002]:

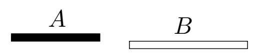
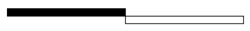
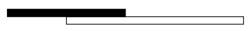
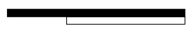
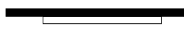
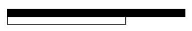
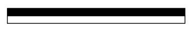
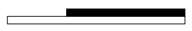
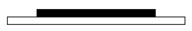
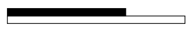
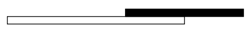
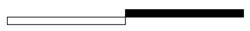
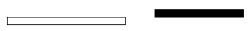
Graphical representation	Relative situations	S−words
	$A \xrightarrow{p\text{ (precede)}} B$	`[{a}{a}{b}{b}]`
	$A \xrightarrow{m\text{ (meet)}} B$	`[{a}{a,b}{b}]`
	$A \xrightarrow{o\text{ (overlap)}} B$	`[{a}{b}{a}{b}]`
	$A \xrightarrow{fi\text{ (finish inverse)}} B$	`[{a}{b}{a,b}]`
	$A \xrightarrow{d\text{ (during inverse)}} B$	`[{a}{b}{b}{a}]`
	$A \xrightarrow{si\text{ (start inverse)}} B$	`[{a,b}{b}{a}]`
	$A \xrightarrow{=\text{ (equal)}} B$	`[{a,b}{a,b}]`
	$A \xrightarrow{f\text{ (finish)}} B$	`[{b}{a}{a,b}]`
	$A \xrightarrow{d\text{ (during)}} B$	`[{b}{a}{a}{b}]`
	$A \xrightarrow{s\text{ (start)}} B$	`[{a,b}{a}{b}]`
	$A \xrightarrow{oi\text{ (overlap inverse)}} B$	`[{b}{a}{b}{a}]`
	$A \xrightarrow{mi\text{ (meet inverse)}} B$	`[{b}{a,b}{a}]`
	$A \xrightarrow{pi\text{ (precede inverse)}} B$	`[{b}{b}{a}{a}]`

Figure 4: The thirteen relative situations between two intervals on a line

$D(p_1,\ldots,p_n) = \sum_{(\widetilde{p_1},\ldots,\widetilde{p_n})\in\mathcal{P}(p_1,\ldots,p_n)} D_n(\widetilde{p_1},\ldots,\widetilde{p_n})$ with

$$\begin{cases} D_1(0) = 1 \\ D_n(p_1,\ldots,p_{n-1},0) = D_{n-1}(p_1,\ldots,p_{n-1}) \end{cases} \quad \text{and}$$

$$\mathcal{P}(p_1,\ldots,p_n) = \left\{ \widetilde{p_1},\ldots,\widetilde{p_n} \in \mathbb{N}^n \mid \begin{array}{ll} \widetilde{p_i} \in \{p_i, p_i - 1\} & \text{if } p_i \neq 0, \\ \widetilde{p_i} \in \widetilde{p_i} = p_i & \text{otherwise} \end{array} \right\} \setminus \{(p_1,\ldots,p_n)\}$$

For instance, for n points, the first terms of the sequence are $D(1) = 1$, $D(1,1) = 3$, $D(1,1,1) = 13$ and for n intervals, the first terms of the sequence are $D(2) = 1$, $D(2,2) = 13$ (Allen's situations), $D(2,2,2) = 409$. The situation shown in Figure 5 is one among the $D(6,6) = 8989$ relative situations.

Recall that a set of n relative situations provides 2^n relative relations. In particular, the 13 situations between two intervals provide $2^{13} = 8182$ relations.

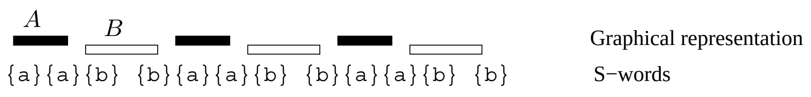


Figure 5: A *alternates* B

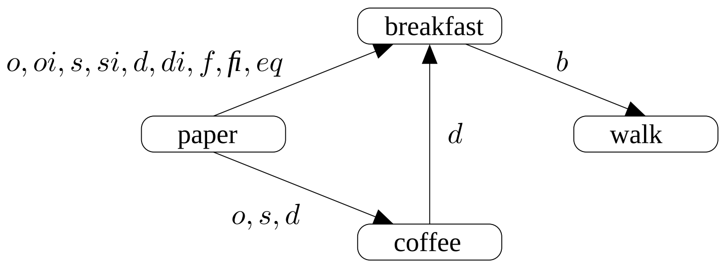


Figure 6: Specification of Fred’s problem

2.1.2 Specifications and Models

Given a set of (temporal) items, a *specification* is a set of partial relations, that is, relations that do not involve all items. A *model* is a (global) situation, that is a situation involving all items. A model satisfies a specification if for each relation of the specification, the restriction of the situation to the items involved in that relation is included in it. There may be several models satisfying the specification. For instance, [van Beek, 1990] provided the following specification, written inside Allen’s glossary (cf. Figure 4) for Fred’s problem presented on Page 2 :
– ’paper’ $\{o, oi, s, si, d, di, f, fi, eq\}$ ’breakfast’
– ’paper’ $\{o, s, d\}$ ’coffee’
– ’coffee’ d ’breakfast’
– ’breakfast’ b ’walk’.
All these relations are binary. There is no relation expressed between ’paper’ and ’walk’, and ’coffee’ and’ walk’. This means that there is no constraint between these items, that is the full relation (containing the thirteen situations) ties them. The full relation is always omitted. There is a natural graphical notation where the vertices represent intervals and the directed edges are labeled with sets of situation names. The resulting graphs are called *(interval) algebra networks*. Figure 6 depicts the algebra network for Fred’s problem.

3 Temporal reasoning

We now address the second problem. Given a specification, several problems can be set up among which:

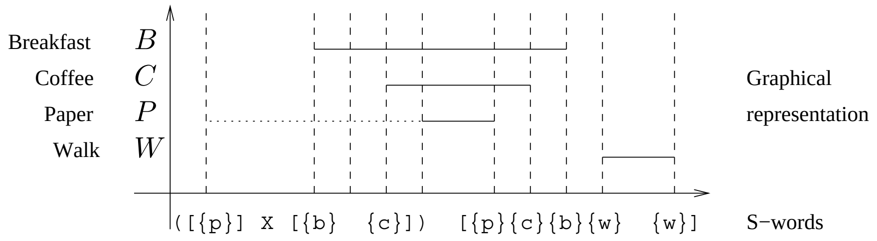


Figure 7: Representation of all models that satisfy the specification of Fred's problem

1. Is there a model satisfying the specification?
2. Find a model satisfying the specification.
3. Find all models satisfying the specification.

Temporal reasoning consists in solving these kind of problems. Figure 7 shows the five models satisfying the specification of Fred's problem given Figure 6. The five solutions are materialized by the dashed part of the segment associated with P which shows the five possible beginnings for P (the intersections with the vertical dashed lines). So to be exact, we should draw five pictures, each showing one of the possible positions for P. Note in the bottom part of the picture the solution in the S-languages formalism; the solution is an S-expression which uses the mix operator to express that there are no constraints between the beginning of P and the the interval starting with the beginning of B and ending with the beginning of C. We shall return to this in Section 5.1.

There are several ways of representing and solving these problems; the most used are first-order logic and modal logic. All these approaches are restricted to binary relations and based on the composition table of the set of corresponding situations. Hence, there is a table for point-point, point-interval, interval-point, interval-interval relations. [Randell et al., 1992] shows how tedious it is to manage such tables.

Temporal relation algebras are based on a unary operator, the *converse*, and a binary operator, the *composition*. The converse operator ($\sim$) is a change of point of view: $R^{\sim}(B, A) = R(A, B)$. In the Interval Algebra, the converse correspondence between situations is indicated by *inverse* (i) as shown in the second column of Figure 4 page 7. In temporal reasoning, intersection and composition are related to two different questions [van Beek, 1990].

Question 1 Let $R_1(A, B)$ and $R_2(A, B)$ be two sets of constraints between items A and B, what is the resulting set of constraints $R_3(A, B)$ between A and B?

Question 2 Given three intervals A, B and C, $R_1(A, C)$, $R_2(C, B)$, two sets of relative constraints between A and C and between B and C, what are $R_3(A, B)$ the constraints between A and B?

Answering the first question requires the *logical And operator* or the *set intersection operator*.The second question requires a transitivity operator based on tables. In Section 5, we show that these two questions are two instantiations of the same problem: the combination of two sets of constraints.

[Vilain et al., 1989] showed that the consistency problem (the existence of a model), and its computationally equivalent minimal labeling problem is NP-hard for intervals but polynomial for points. This computational complexity has motivated the study of heuristics based on *tractable* sub-algebras of the Interval Algebra. The best known sub-algebra are the convex one [Nökel, 1990], which contains 83 elements, the pointizable sub-algebra [Ladkin and Maddux, 1994] which contains 189 elements and the ORD-Horn sub-algebra [van Beek and Cohen, 1990] which contains 868 elements. In [Krokhin et al., 2003] a characterization of all such sub-algebras has been provided.

The S-words approach is not directly concerned with these sub-algebras, because our reasoning does not rely on such networks. Nevertheless, we are concerned with computational complexity. We are currently investigating sub-classes of S-languages which allow polynomial time reasoning. [Dubois and Schwer, 2000] describes the convex class in terms of S-languages.

4 The S-languages formalism

In this section, we present in details the bases of the S-languages formalism.

4.1 Letters and S-letters

Each temporal item E is represented by a *letter* $\mathtt{e}$. By α, we denote the *alphabet* of all letters. For implementation issues, it is supposed to be linearly ordered according to the order of the letters in their enumeration.

For instance, $\alpha = \{\mathtt{a,b,c}\}$ yields $\mathtt{a} < \mathtt{b} < \mathtt{c}$. $\#\alpha$ denotes the cardinality of α. An *S-letter* is a non-empty subset of α. It defines synchronization points between events. For instance, $\{\mathtt{a,c}\}$ is an S-letter meaning that the extremity point (or point) $\mathtt{a}$ and the extremity point (or point) $\mathtt{c}$ occur simultaneously. By $S_\alpha = \mathcal{P}(\alpha) \setminus \{\emptyset\}$, we denote the set of S-letters whose *underlying* alphabet is α. Note that $\#S_\alpha = 2^{\#\alpha} - 1$. For instance, $S_{\{\mathtt{a,b,c}\}} = \{\{\mathtt{a}\},\{\mathtt{b}\},\{\mathtt{c}\},\{\mathtt{a,b}\},\{\mathtt{a,c}\},\{\mathtt{b,c}\},\{\mathtt{a,b,c}\}\}$. Note that the meaning of a single S-letter is very ambiguous without its context.

4.2 S-words

An *S-word* is a finite sequence of S-letters, in other words an element of $(S_\alpha)^*$. We surround the sequences of the S-letters of an S-word by brackets " [", "] ". The alphabet $\alpha(w)$ of an S-word w is the set of letters appearing in its S-letters.

The *Parikh value* of a letter $l \in \alpha(w)$, is its number of occurrences in w. The *Parikh vector* of an S-word w (denoted by $\vec{p_w}$) is the vector of $\mathbb{N}^{\#\alpha}$ whose i^{th} coordinate is the Parikh value of the i^{th} letter. With a Parikh vector $\vec{p}$, we associate the *Parikh map* P which maps every letter to its Parikh value. The *Parikh alphabet* of an S-word w is the pair $\langle \alpha(w), \vec{p_w} \rangle$. Let $\langle \alpha, \vec{p} \rangle$ and $\langle \alpha', \vec{p'} \rangle$ be two Parikh alphabets. They are said to be *compatible* if $\forall l \in \alpha \cap \alpha'$, $P(l) = P'(l)$.

Two compatible Parikh alphabets $\beta_1 = \langle \alpha_1, \vec{p_1} \rangle$ and $\beta_2 = \langle \alpha_2, \vec{p_2} \rangle$ can be joined into the Parikh alphabet $\beta = \beta_1 \;\mathtt{J}\; \beta_2 = \langle \alpha, \vec{p} \rangle$ where $\alpha = \alpha_1 \cup \alpha_2$[6] and $\forall l \in \alpha$,

$$\begin{array}{ll} P(l) = P_1(l) = P_2(l) & if\ l \in \alpha_1 \cap \alpha_2 \\ P(l) = P_1(l) & if\ l \in \alpha_1 \setminus \alpha_2 \\ P(l) = P_2(l) & if\ l \in \alpha_2 \setminus \alpha_1 \end{array}$$

In words, the letters of α is the union of the letters of β_1 and β_2. The Parikh values are the original ones and there is no conflict as α_1 and α_2 are compatible (same Parikh value for identical letters).

Example 1. Let $\alpha = \{\mathtt{a,b,c,d}\}$ and $w_3 = \mathtt{[\{a\}\{a,b\}\{a,c\}\{a,b,c\}]}$.
$\vec{p_{w_3}} = (4, 2, 2, 0)$. $P_{w_3}(\mathtt{a}) = 4$, $P_{w_3}(\mathtt{b}) = 2$, $P_{w_3}(\mathtt{c}) = 2$, $P_{w_3}(\mathtt{d}) = 0$.
Let $\alpha' = \{\mathtt{a,e}\}$. The Parikh alphabets $\langle \alpha, (4, 2, 2, 0) \rangle$ and $\langle \alpha', (4, 3) \rangle$ are compatible.
$\langle \alpha, (4, 2, 2, 0) \rangle \;\mathtt{J}\; \langle \alpha', (4, 3) \rangle = \langle \alpha \cup \alpha', (4, 2, 2, 0, 3) \rangle$.

4.3 S-languages

An *S-language* is a set of S-words. The set of all possible S-words with identical Parikh alphabet β is called the *S-universe* of β and is denoted by $\mathcal{U}(\beta)$. It is a finite language. An *S-language over a Parikh alphabet* β is a subset of $\mathcal{U}(\beta)$.

Given an S-language L, the alphabet $\alpha(L)$ of L is the set of letters of the S-words of L.

A S-language can be described either in *extension*, *i.e.* by giving the list of its S-words (this is possible in the finite case only: finite language of finite S-words) or by expressions over S-languages using operators. Operations and expressions over S-languages will be presented in Section 4.4.

Suppose for instance that we have two independent lasting items A and B, each occurring once. They are represented by the S-words `[{a}{a}]` and `[{b}{b}]` respectively. The S-universe $\mathcal{U}(\langle \{\mathtt{a,b}\}, (2, 2) \rangle)$ contains exactly the 13 S-words presented in Figure 4 Page 7. We shall see in Section 4.4.3 that any S-universe over a Parikh alphabet can be represented by a simple expression over S-languages.

Any S-language over a Parikh alphabet β corresponds to a specific set of temporal constraints over the corresponding temporal items. Conversely, each set of temporal constraints over a Parikh alphabet corresponds to a specific S-language. If the Parikh vector is $(2, 2)$ there are 2^{13} possible S-languages; the whole set represents the absence of constraint; the empty set represents incompatible constraints.

[6] The order in α is both compatible with the orders in α_1 and α_2, if $x \in \alpha_1$ and $y \in \alpha_2 \setminus \alpha_1$, then $x < y$.

Example 2. For instance, if we add the constraints that B must start strictly after A and end after or at the same time as A, we get the following S-language with 4 possibilities: $L_1 = \{$ `[{a}{b}{a,b}]`, `[{a}{a}{b}{b}]`, `[{a}{a,b}{b}]`, `[{a}{b}{a}{b}]` $\}$.

S-languages are not always restricted to a finite S-universe. In [Schwer, 2007], infinite S-words are used to deal with execution traces in distributed systems.

4.4 Operations on S-languages, S-expressions

4.4.1 Classical operations on languages

S-languages are a special case of formal languages. Consequently, all classical operations on formal languages apply [Rozenberg and Salomaa, 1996]. In particular, the Boolean operations (union, intersection, complement), concatenation, mirror are defined in the usual way considering that S-letters are the letters of the S-words. In the classical framework, letters are basic objects which cannot be decomposed. In S-languages, the letters of the S-words are S-letters, *i.e.* sets of letters which we may want to compose or decompose. Expressions over S-languages will be referred to as S-expressions[7]. The classical projection would be to project over a sub-alphabet of S-letters which erases S-letters.

4.4.2 S-projection

In the S-language framework, we define the S-projection over a sub-alphabet of letters which erases letters inside the S-letters of an S-word. The same extension can be considered for morphisms and inverse morphisms.

The *S-projection* of an S-word w over the alphabet α, denoted by $w_{|\alpha}$ is the S-word obtained by erasing from w all occurrences of letters which are not in α and then removing every S-letter which has become empty. The S-projection can be computed in polynomial time with regards to the sum of the size of the S-word and the length of the alphabet.

Example 3. Let $w =$ `[{a,c}{a,b}{c,d}{a,b,c}]` and $\alpha = \{$`a,b`$\}$.
$w_{|\alpha} =$ `[{a}{a,b}{a,b}]`.

The S-projection of an S-language is the set of the S-projections of its S-words.

4.4.3 The join operation

Consider two S-languages L_1, L_2. Each S-language L_i represents temporal constraints. *Joining* the two languages L_1 and L_2 corresponds to constraining with the union of the constraints of both languages. The join operation will be denoted by the symbol `J`. The formal definition will be given Page 12. The join is the ”magic” operator which answers our two questions.

[7] not to be confused with `Lisp` Sexpressions (`Sexpr`).

Example 4. Let $L_1 = \{$ `[{a}{b}{a,b}],[{a}{a}{b}{b}],[{a}{a,b}{b}]` $\}$ of Example 2. The language $L_2 = \{$ `[{a}{a,c}{c}]` $\}$ can be described by the constraint "C starts when A stops". L_1 `J` L_2 yields the following S-language
`{ [{a}{b}{a,b,c}{c}], [{a}{a,c}{b,c}{b}], [{a}{a,c}{b}{b,c}],`
`[{a}{a,c}{b}{b}{c}], [{a}{a,c}{b}{c}{b}], [{a}{a,c}{c}{b}{b}],`
`[{a}{a,b,c}{b,c}], [{a}{a,b,c}{b}{c}],[{a}{a,b,c}{c}{b}] }`.

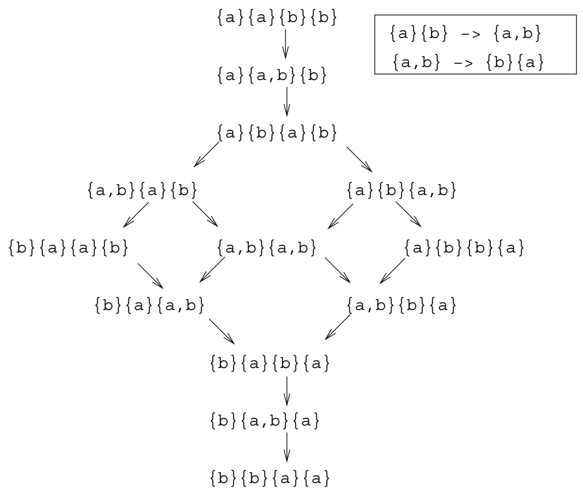
We can see that the constraints "C starts when A stops" is satisfied and that for the ending of C and beginning of B all possibilities are considered.

There exist two special cases for the join operation: the first case occurs when the alphabets of the two languages are identical, then the join corresponds to the intersection of the two languages; the second case occurs when the alphabets are disjoint and is described below.

In the case of disjoint alphabets, the join operation is a kind of *shuffle* that we call *mix* and denote by `X`: it considers all possibilities of ordering independent letters. In the case where $L_1 =$`[{a}{a}]` and $L_2 =$`[{b}{b}]`, the S-language corresponding to L_1 `J` L_2 can be obtained by applying the two rewrite rules

```
        {a}{b} -> {a,b}            {a,b} -> {b}{a}
```

on the concatenation of L_1 and L_2 which is `[{a}{a}{b}{b}]`. The lattice (shown in Figure 8) obtained by applying the rewrite rules contains the 13 S-words of L_1 `X` L_2. Note that these S-words are the same as the ones in Figure 4 Page 7. This principle



$$\begin{tikzcd}
 & & \{a\}\{a\}\{b\}\{b\} \arrow[d] & & \boxed{\begin{array}{l}\{a\}\{b\}\ \text{->}\ \{a,b\}\\ \{a,b\}\ \text{->}\ \{b\}\{a\}\end{array}} \\
 & & \{a\}\{a,b\}\{b\} \arrow[d] & & \\
 & & \{a\}\{b\}\{a\}\{b\} \arrow[dl] \arrow[dr] & & \\
 & \{a,b\}\{a\}\{b\} \arrow[dl] \arrow[dr] & & \{a\}\{b\}\{a,b\} \arrow[dl] \arrow[dr] & \\
\{b\}\{a\}\{a\}\{b\} \arrow[dr] & & \{a,b\}\{a,b\} \arrow[dl] \arrow[dr] & & \{a\}\{b\}\{b\}\{a\} \arrow[dl] \\
 & \{b\}\{a\}\{a,b\} \arrow[dr] & & \{a,b\}\{b\}\{a\} \arrow[dl] & \\
 & & \{b\}\{a\}\{b\}\{a\} \arrow[d] & & \\
 & & \{b\}\{a,b\}\{a\} \arrow[d] & & \\
 & & \{b\}\{b\}\{a\}\{a\} & &
\end{tikzcd}$$

Figure 8: Lattice of the mix operation

generalizes to any Parikh alphabet β. Let $\alpha = \{\mathtt{a},\mathtt{b},\cdots\}$. If $\beta = \langle \alpha, (v_{\mathtt{a}}, v_{\mathtt{b}}, \cdots)\rangle$ then $\mathcal{U}(\beta) =$ `[{a}...{a}] X [{b}...{b}] X ...` where each S-word of the mix contains as many letters l as the Parikh value of l. A mix expression is a compact way of representing an S-universe, whose cardinality is $D(v_{\mathtt{a}}, v_{\mathtt{b}}, \cdots)$.

The mix operation is commutative and associative. The mix operation is typically exponential with regard to the sum of the sizes of the mixed S-words because the size of the result is exponential. However, in practice, we shall try to avoid as much as possible computing mixes, keeping the exponential complexity buried in mix expressions. Therefore the mix operation is a necessary tool to handle S-languages.

The main reason to differentiate the mix from the join is that it is handled in a completely different way.

In the general case, the alphabets have a non-empty intersection ($\alpha(L_1) \cap \alpha(L_2) \neq \emptyset$). The join operation is first defined on S-words. Let f and g be two S-words. If $\alpha(f) \cap \alpha(g) = \emptyset$ then $f \mathtt{~J~} g = f \mathtt{~X~} g$ as defined above. Otherwise, let $\alpha = \alpha(f) \cap \alpha(g) \neq \emptyset$. If the projections $f_{|\alpha}$ and $g_{|\alpha}$ differ then the constraints inherent to the two S-words are incompatible and $f \mathtt{~J~} g = \{\}$. Otherwise, the S-words are compatible and $f \mathtt{~J~} g$ is the S-language containing all S-words h written over α which satisfy $h_{|\alpha(f)} = f$ and $h_{|\alpha(g)} = g$. Let for instance,
$f =$ `[{a,c}{a,b}{c,d}{a,b,c}]` and
$g =$ `[{e}{a,e,f}{e}{a,b}{f}{a,b,f}{e}]`.
Then $\alpha = \{$ `a,b` $\}$, $f_{|\alpha} =$ `[{a}{a,b}{a,b}]` $= g_{|\alpha}$ and

```
f J g = { [{e}{a,c,e,f}{e}{a,b}{c,d,f}{a,b,c,f}{e}],
          [{e}{a,c,e,f}{e}{a,b}{c,d}{f}{a,b,c,f}{e}],
          [{e}{a,c,e,f}{e}{a,b}{f}{c,d}{a,b,c,f}{e}] }
```

However, we can give a more compact representation using the mix operation:

```
[{e}{a,c,e,f}{e}{a,b}] . ([{c,d}] X [{f}]) . [{a,b,c,f}{e}]
```

The complexity of the join operation is something in between the complexity of the mix (exponential) and the intersection (polynomial). It is always interesting to compute the projections because if they differ we obtain the result of the join (the empty language) in polynomial time.

The art of computing a join operation is to recognize common sub-S-words that can be factored out using a mix-operation; this avoids computing the extension of this mix operation.

The join operation extends to languages: the join of two S-languages is the union of the joins of an S-word of the first language and an S-word of the second. The join operator on S-languages is associative and commutative. A description of an algorithm for joining S-words can be found in [Schwer, 2004]. Our implementation provides both a recursive and an iterative version of it.

5 Temporal reasoning with S-languages

In the S-words framework, we do not need the notion of ”point of view” and the converse operator (see Page 3) is the identity operator on S-words. This makes our representation more compact. ”A overlaps B” and ”B is overlapped by A” correspond to the unique S-word `[{a}{b}{a}{b}]`. Let us note that the mirror on S-languages corresponds to an inversion of the time direction : the mirror of `[{a}{b}{a}{b}]` is `[{b}{a}{b}{a}]` which corresponds to ”B overlaps A” and ”A is overlapped by B”. The composition of two relations corresponds to a join operation followed by a projection.

In the S-languages framework, we can model both Question 1 and Question 2 of Page 9 by first joining the two S-languages describing the two relations, and then S-projecting on their common Parikh alphabet.

Let us denote by $L_i(X, Y)$, an S-expression corresponding to $R_i(X, Y)$, where $\beta(X, Y)$ is the Parikh alphabet associated to the temporal items X and Y.

Given a set of relations $R_1(U, V)$ between two items U and V and a set of relations $R_2(X, Y)$ between two items X and Y, the set of possible relations $R_3(U, Y)$ between U and Y is given by the general formula:

$$(L_1(U, V) \ \mathtt{J} \ L_2(X, Y))_{|\beta(U,Y)}$$

Questions 1 and 2 are just a particular case of this more general problem. Question 1 is solved by

$$(L_1(A, B) \ \mathtt{J} \ L_2(A, B))_{|\beta(A,B)} = L_1(A, B) \ \mathtt{J} \ L_2(A, B)$$

and Question 2 by

$$(L_1(A, C) \ \mathtt{J} \ L_2(C, B))_{|\beta(A,B)}$$

Relation algebra deals with binary relations only. But there is no such a limitation in the S-language framework.

More generally, a specification consists of a Parikh alphabet β describing a set of temporal items and a set of n S-expressions $S_1, \ldots, S_n$ over β, each one specifying a set of temporal constraints between some items (not necessarily restricted to pairs). Let $S = S_1 \ \mathtt{J} \ S_n \ \mathtt{J} \ \mathcal{U}(\beta)$ and L be the S-language corresponding to S.

The specification has solutions if and only if the S-language L represented by the S-expression $S = S_1 \ \mathtt{J} \ S_2 \ldots S_n$ is nonempty. So, all three questions of Page 8 are answered just by computing the S-language corresponding to S. One model satisfying the specification is any S-word belonging to L. All the models are given by the S-words of L.

In finite S-universes, as we can compute all the S-operations over S-words and S-languages, it would be theoretically possible to compute in extension the L_is and to compute $L = L_1 \ \mathtt{J} \ L_2 \ldots L_n$. But this is not tractable in general because some of the intermediate S-languages can be very large. The key idea is to simplify S until it becomes reasonable to compute the final S-language or until it becomes possible to conclude without computing the final S-language whether it is empty or not. We should especially avoid evaluating mix expressions which represent in a compact way partial S-universes which can be very large if represented by the corresponding S-language. For instance, in an expression like `([[{a,b,e}] . ([{a}{b}] X [{e}])) X [{c}{c}] X [{d}{d}],`

it is uninformative and harmful to evaluate the part `[{c}{c}] X [{d}{d}]`; this part expresses all possibilities of placing the temporal items C and D with regard to the others.

Finding simplifications and proving they are correct is a difficult domain which is not completely explored. Subsection 5.3 gives hints in that direction. We can also consider the problem of verifying whether an S-word w satisfies the specification. To solve this problem, we just compute w `J` S_i for each i; if every resulting corresponding S-language is non empty, w satisfies the specification.

5.1 Fred's example

We shall now solve Fred's problem in the S-languages framework. Let us now translate Fred's example presented on Page 2. The graphical specification was shown in Figure 6. There are four items associated with four temporal intervals: eating breakfast B, reading the paper P, going for a walk W and drinking coffee C. The S-universe is $\mathcal{U}(\langle\{\texttt{b,c,p,w}\},(2,2,2,2)\rangle)$.
The S-expression $S_{(B,P)}$ = `([{b}] X [{p}]).([{b}] X [{p}]` specifies the labeled edge between P and B.
$S_{(B,C)}$ = `[{b}{c}{c}{b}]` specifies the labeled edge between B and C.
$S_{(B,W)}$ = `[{b}{b}{w}{w}]` specifies the labeled edge between B and W.
$S_{(C,P)}$ = `([{c}] X [{p}]).[{p}{c}]` specifies the labeled edge between C and P.
To solve the problem we simplify the S-expression
$S = S_{(B,P)}$ `J` $S_{(B,C)}$ `J` $S_{(B,W)}$ `J` $S_{(C,P)}$
which gives the S-expression
S' = `([{p}]X[{b}{c}]).[{p}{c}{b}{w}{w}]`
and finally we may compute the corresponding S-language which contains five S-words which means that we have the five models depicted in Figure 7. However, it is not necessary to compute S' in order to find the cardinality of the corresponding S-languages.

5.2 The example of the trains

We formalize the problem in the S-languages framework. For each train, we consider the temporal item corresponding to the time during which the train remains at the platform.

We have 6 lasting temporal items. Our alphabet is $\alpha = \{\texttt{a,b,c,d,e,f}\}$, one letter for each train. The S-universe is $\mathcal{U}(\langle\{\texttt{a,b,c,d,e,f}\},(2,2,2,2,2,2)\rangle)$.
It contains $D(2,2,2,2,2,2) = 308682013$ S-words. The five constraints can be expressed by the following five S-expressions:

1. S_1 = `[{a,b,e}] . ([{a}] X [{b}] X [{e}])`
2. S_2 = `([{a}] X [{b}]) . [{a}{b}]`

3. $S_3 =$ `(([{a}] X [{c}{c}]) . [{a}{d}{d}]) U`
   `(([{a}] X [{c}]) . [{a,c}{d}{d}])`
4. $S_4 =$ `[{b}{b,d,f}] . ([{f}] X [{d}])`
5. $S_5 =$ `([{e}] X [{d}]) . [{d,e}]`

To solve the problem we must first simplify the S-expression
$S = S_1$ J S_2 J S_3 J S_4 J S_5. As we already mentioned, simplifying S-expressions is non a trivial matter. A whole article is to be written to address this problem. The next section gives a few hints in that direction.

5.3 Simplifying S-expressions

The first kind of simplifications results from classical properties of the operators like associativity, commutativity, idempotence and distributivity. The other simplifications concern the join operation or its special cases (mix, intersection). For instance, the intersection of two languages with disjoint alphabets is empty; the join of a language with its S-universe is the language itself.

For our trains example of Section 5.2, the tool `SLS` described in Section 6 is able to simplify the S-expression which evaluates to an S-language containing 24 S-words of length between 5 and 7. The final language can be written using mix as:

```
S = ([{c}{c}] X [{a,b,e}]).[{a}].[{b,d,f}].([{f}] X [{e,d}]) U
      ([{a,b,e}] X [{c}]).[{a,c}].[{b,d,f}].([{f}] X [{e,d}])
```

In order to solve our problem, we have to recall the good interpretation of what this S-language depicts (the possible relationships between the periods where trains are stopped at a platform), then to find inside the S-language `S` an S-word which minimizes the meeting or interleaving between these periods. The first choice is to take
`([{c}{c}{a,b,e}])`
from the left sub-S-expression `([{c}{c}] X [{a,b,e}])` which isolates the train `C`. The new S-expression is
`S'=[{c}{c}{a,b,e}{a}{b,d,f}].([{f}] X [{e,d}])`
and contains only 3 S-words. First, `C` stops and leaves, then `A`, `B`, `E` arrive all at the same time, therefore we need at least three tracks. When `D` and `F` arrive, `A` has already left but `B` being currently leaving we need one more track. The answer of the problem is then that a minimum of 4 tracks is necessary; 4 tracks are also sufficient for all S-words of `S'`.

5.4 Directed S-expressions

An S-language which can be represented by a partial order between the points (or extremity points of intervals) is said to be *directed* and so is an S-expression representing a directed S-language. A directed S-language can be represented as a directed acyclic graph (DAG) whose nodes are S-letters. Joining two directed S-expressions can be done

in linear time with regards to the sum of the sizes of the DAGs by joining the DAGs. For instance, consider the two directed S-expressions
$S_1 =$ `[{a}{b}] . ([{a,d}] X [{b}])` and $S_2 =$ `[{a}{c}] . [{a,c}]`.
The corresponding DAGs and the DAG resulting from the join are shown in Figure 9.

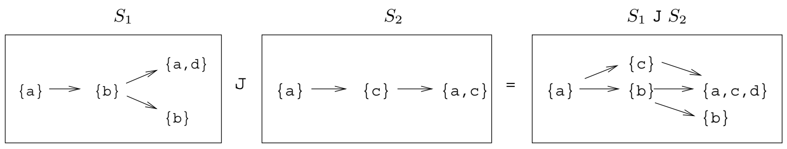


Figure 9: DAGs for S_1, S_2 and for S_1 J S_2.

A possible S-expression for the resulting DAG is
`([{a}] . (([{b}] . ([{b}] X [{a,d,c}])) J [{c}{a,d,c}]))`.

To help the reader understand what a directed S-language is, we now give example of a non-directed S-languages: $L_3 = \{$ `[{a}{a}{b}{b}], [{a}{b}{a}{b}]` $\}$ and $L_4 = \{$ `[{a}{a}{b}{b}], [{a}{a,b}{b}]` $\}$ are not directed but can be described as a union of directed S-languages as individual S-words are directed S-languages.

Directed S-languages play the same role as the polynomial sub-classes of Allen's algebra. They help avoiding combinatorial explosion in many cases.

6 Implementation of S-languages

It will not take long to justify the choice of the Common Lisp language to implement the theory of S-languages: the domain is typically symbolic as opposed to numeric; the data are highly hierarchical which justifies an object-oriented language; in addition, multiple inheritance is very useful for factoring properties and associated methods for simplifying S-expressions.

6.1 Implementation of basic objects

The basic `SLS` objects are letters `(letter)`, marked letters `(mletter)`, S-letters `(sletter)`, alphabets for all the different kinds of letters `(alphabet, malphabet)`, S-words `(sword)`.

To prevent combinatorial explosion we use the well-known technique of *hash-consing*: each element of each object category is represented by a unique Lisp object; there is a list for each category of object; the objects are stored in the list corresponding to its category. When the creation of an object is required, a look-up is done in the corresponding list; if an object with equal components (in the `eq` sense) is found such object is returned; otherwise a new object is constructed and stored in the list. Here is the example of the `mletter` case.

```
(defmethod make-mletter ((string string) &optional (mark 0))
  (let* ((letter (make-letter string))
         (name (name letter)))
    (or (find-object name (mletters *spec*)
                     :test (lambda (name mletter)
                              (and (eq name (name mletter))
                                   (= mark (mark mletter)))))
        (let ((mletter (make-instance 'mletter :letter letter
                                      :mark mark)))
          (setf (mletters *spec*)
                (append (mletters *spec*) (list mletter)))
          mletter))))
```

This technique has also the advantage that SLS basic objects are eq-comparable which improves time performance.

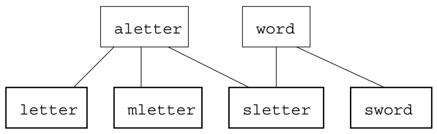


Figure 10: Classes for basic SLS objects

The hierarchy of the classes describing basic SLS objects is presented in Figure 10. Note that an S-letter, being a sequence of letters, is itself a word (but not an S-word).

6.2 Implementation of S-expressions

The class alanguage contains all objects which describe languages. A language can be represented by its set of words (language, word) or by an expression. An expression is defined recursively: it is either a concrete language or an expression with an operator and whose arguments are expressions. Note the use of the mixin classes to capture properties which help simplifying expressions. For instance, the primary method clean-args normalizes the arguments of an associative S-expression. The secondary methods complete this task according to the other properties of an operator. For instance, if the operator is idempotent, we can remove duplicated or equivalent arguments.

```
(defmethod clean-args ((lexpr assoc-lexpr)) ...)
(defmethod clean-args :before ((lexpr fold-mixin))
  (setf (args lexpr)
        (remove-duplicates (args lexpr) :test #'equivalent)))
```

Some directed S-expressions are easily recognizable. When a join is to be done between such expressions, they are lazily transformed into their DAG counterparts and the join is performed efficiently on the DAGs (as described in Section 5.4).

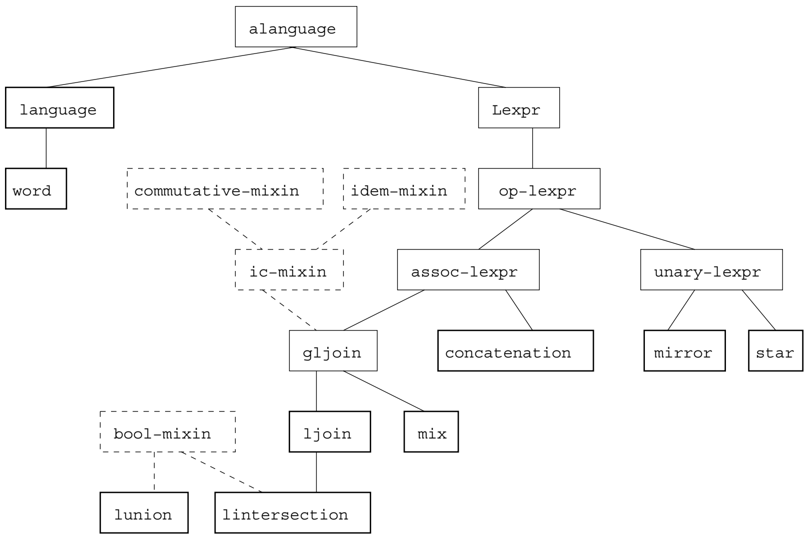


Figure 11: Class hierarchy for representing S-languages

6.3 Specifications for `SLS`

`SLS` handles a set of specifications that can be loaded interactively. A specification consists of a signature, possibly a set of variables, followed by a list of `SLS` objects. `SLS` objects are S-words, S-expressions, S-languages, or Problems (set of S-expressions which correspond to constraints). In a same specification, one stores objects from a common S-universe.

Figure 12 shows an example of such a specification. That specification contains the train problem of Section 5.2. It also shows how to specify S-words, S-expressions or S-languages in extension.

6.4 The graphical interface

A graphical user interface helps the user to load his/her data (S-words, S-expressions, S-languages) and apply operations on it. It is written using the `McCLIM` [Strandh and Moore, 2002] system which is the free implementation of the `CLIM` specification. A snapshot of the `SLS` window after loading the `train.txt` specification is shown in Figure 13. All the commands are either accessible from the command line in the top window or from menus, classified according to the type of object they operate on. Here we have applied the command `Solve` (also in the `Problem` menu) which transforms the set of constraints of the problem into a (when possible) simplified S-expression which becomes the current S-expression. Next we have applied the `Slanguage Sexpr` command (also in the `Sexpr` menu) which computes the S-language corresponding to the current S-expression and invoked the `Cardinality Slanguage`

```
Problem trains
 ([{a,b,e}] . ([{a}] X [{b}] X [{e}]))
 (([{a}] X [{b}]) . [{a}{b}])
 ((([{a}] X [{c}{c}]) . [{a}{d}{d}]) U
  (([{a}] X [{c}]) . [{a,c}{d}{d}]))
 ([{b}{b,d,f}] . ([{f}] X [{d}]))
 (([{e}] X [{d}]) . [{d,e}])

Sword [{c}{c}{a,b,e}{a}{b,d,f}{e,d}{f}]
Sexpr ([{a}{a}] X [{b}{b}])
Sexpr ([{a,b}] J [{b_1,c}])
Slanguage L {[{a}{b}{a,b}],  [{a}{a}{b}{b}], [{a}{a,b}{b}],
             [{a}{b}{a}{b}]}
```

Figure 12: Example of an SLS specification

command (also in the SLanguage menu) which prints the cardinality of the current S-language. Finally, with the Membership To Slanguage, we verify that the current S-word belongs to the current S-language. The final look of the window is shown in Figure 14.

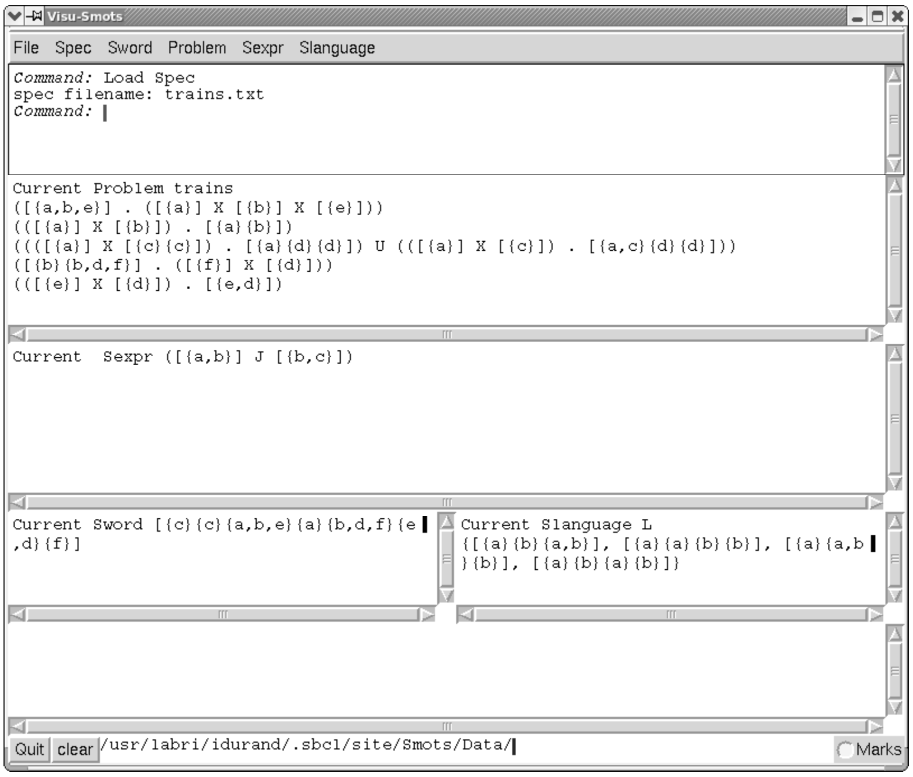
SLS contains altogether 6000 lines of Common Lisp of which around 1200 correspond to the graphical interface. On the project Page [Durand, 2008], one can find a description of the project, a User's Manual, an archive with the latest source and executable files for a few architectures.

7 Related works and perspectives

7.1 Related works

The S-languages framework is based on two fundamental relations: precedence and simultaneity. These relations are essential for modeling distributed systems. In the context of the famous *Allocation Problem*, the three important properties (invariance, aliveness and precedence) were studied inside the modal logic framework by Manna and Pnueli [Manna and Pnueli, 1981]. In [Schwer, 2004], these properties are translated into the S-languages framework. Other tools known to deal with parallelism and concurrency, such as Petri nets [Peterson, 1983] and pomsets [Pratt, 1986] are related to S-languages.

A Petri net is a graphical and mathematical tool for describing and studying concurrent, asynchronous, distributed, parallel, non deterministic and/or stochastic systems and are used as a visual-communication aid similar to flow charts, block diagrams and networks. They are commonly used to define resource allocation problems. Graphically speaking, a Petri net is a finite ordered bipartite directed graph. There are two types of nodes: places (for states or resources), which can contain tokens, and transitions (for actions or changes). Edges are oriented and labeled by integers. The origin (resp. extremity) of an edge is called the *source* (resp. *target*) of the edge. A *marking* of the Petri net is a distribution of tokens in the places. A transition is said to be firable if



```
Command: Load Spec
spec filename: trains.txt
Command:
```

```
Current Problem trains
([{a,b,e}] . ([{a}] X [{b}] X [{e}]))
(([{a}] X [{b}]) . [{a}{b}])
((([{a}] X [{c}{c}]) . [{a}{d}{d}]) U (([{a}] X [{c}]) . [{a,c}{d}{d}]))
([{b}{b,d,f}] . ([{f}] X [{d}]))
(([{e}] X [{d}]) . [{e,d}])
```

```
Current  Sexpr ([{a,b}] J [{b,c}])
```

```
Current Sword [{c}{c}{a,b,e}{a}{b,d,f}{e
,d}{f}]
```

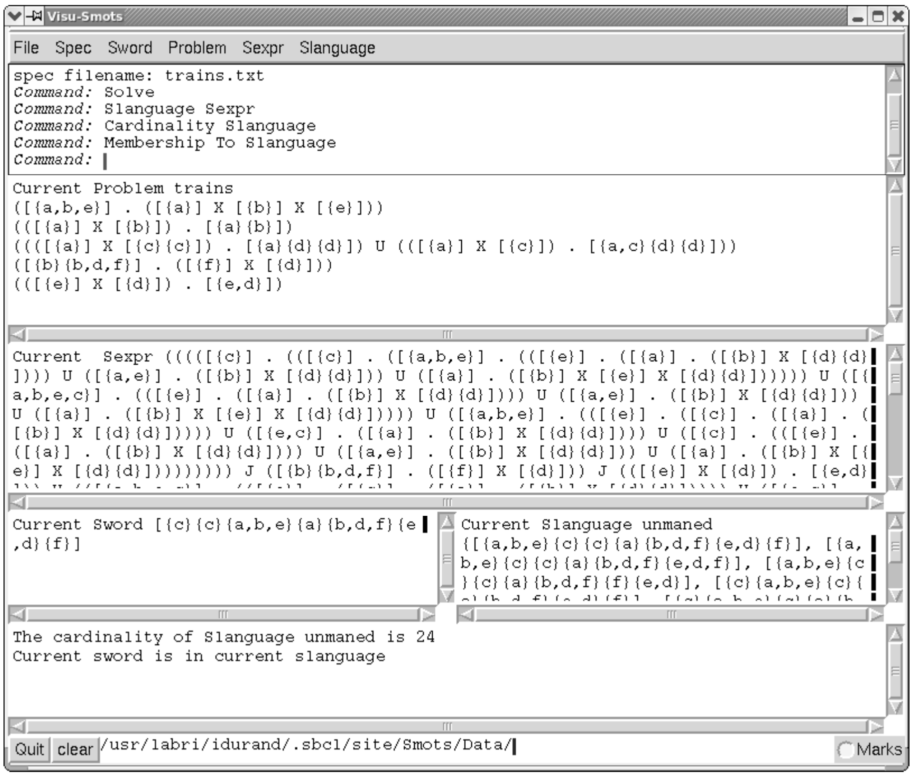
```
Current Slanguage L
{[{a}{b}{a,b}], [{a}{a}{b}{b}], [{a}{a,b
}{b}], [{a}{b}{a}{b}]}
```

```
/usr/labri/idurand/.sbcl/site/Smots/Data/
```

Figure 13: First snapshot of `SLS`

the source of each of its incoming edges contains at least as many tokens as indicated on the edge. The firing of a transition first removes from each of its source places as many token as the number labeling their connecting edge and then adds to each of its target places as many tokens as the number labeling their connecting edge. The possible behaviors of the system that the Petri net models can then be described by words written on the alphabet of transitions as the set of sequences of firable transitions. This set is a language, because Petri nets were built to model asynchronous system. Petri nets have been enriched with temporal annotations to enforce synchronicity and are compared with the constraint satisfaction problem in [Mancel et al., 2002]. [Schwer, 2002] describes a natural correspondence between Petri nets and S-languages by introducing the notion of an S-language of transitions associated to a Petri net.

For modeling concurrency processes, some other models have been developed, based on partial order on multisets (pomsets) in order to deal with multiple occurrences of the same actions. In this framework, the same action can be executed concurrently



```
spec filename: trains.txt
Command: Solve
Command: Slanguage Sexpr
Command: Cardinality Slanguage
Command: Membership To Slanguage
Command:
```

```
Current Problem trains
([{a,b,e}] . ([{a}] X [{b}] X [{e}]))
(([{a}] X [{b}]) . [{a}{b}])
((([{a}] X [{c}{c}]) . [{a}{d}{d}]) U (([{a}] X [{c}]) . [{a,c}{d}{d}]))
([{b}{b,d,f}] . ([{f}] X [{d}]))
(([{e}] X [{d}]) . [{e,d}])
```

```
Current  Sexpr (((([{c}] . (([{c}] . ([{a,b,e}] . (([{e}] . ([{a}] . ([{b}] X [{d}{d}
]))) U ([{a,e}] . ([{b}] X [{d}{d}])) U ([{a}] . ([{b}] X [{e}] X [{d}{d}]))))) U ([{
a,b,e,c}] . (([{e}] . ([{a}] . ([{b}] X [{d}{d}]))) U ([{a,e}] . ([{b}] X [{d}{d}]))
U ([{a}] . ([{b}] X [{e}] X [{d}{d}])))) U ([{a,b,e}] . (([{e}] . ([{c}] . ([{a}] . (
[{b}] X [{d}{d}])))) U ([{e,c}] . ([{a}] . ([{b}] X [{d}{d}]))) U ([{c}] . (([{e}] .
([{a}] . ([{b}] X [{d}{d}]))) U ([{a,e}] . ([{b}] X [{d}{d}])) U ([{a}] . ([{b}] X [{
e}] X [{d}{d}])))))))) J ([{b}{b,d,f}] . ([{f}] X [{d}])) J (([{e}] X [{d}]) . [{e,d}
```

```
Current Sword [{c}{c}{a,b,e}{a}{b,d,f}{e
,d}{f}]
```

```
Current Slanguage unmaned
{[{a,b,e}{c}{c}{a}{b,d,f}{e,d}{f}], [{a,
b,e}{c}{c}{a}{b,d,f}{e,d,f}], [{a,b,e}{c
}{c}{a}{b,d,f}{f}{e,d}], [{c}{a,b,e}{c}{
```

```
The cardinality of Slanguage unmaned is 24
Current sword is in current slanguage
```

```
/usr/labri/idurand/.sbcl/site/Smots/Data/
```

Figure 14: Second snapshot of SLS

by different actors. Occurrences of the same action are thus not linearly ordered like ours. Nevertheless, S-languages seem to be able to describe pomsets by associating an alphabet based on events and actors. More work has to be done in that direction.

Other formalisms propose expressive and compact representations which are more easily handled than binary relations. [Mörchen, 2006], for instance, proposes the Time Series Knowledge Representation. However they address different problems, in particular mining temporal data. For this purpose, they also need quantitative information which we do not address here.

7.2 Conclusion

Temporal items and temporal constraints between them are a crucial matter in many domains (artificial intelligence, linguistics, music,...). Making our software really usable in applications requires work in two directions.

The problem of constraint satisfaction is intrinsically exponential [Krokhin et al., 2003].In S-languages, the mix operation is a way to avoid combinatorial explosion in some cases. For the other cases, and in order to minimize the risk of combinatorial explosion, theoretical work must be done for better simplifying S-expressions before calculating in extension the corresponding S-languages. When we cannot avoid combinatorial explosion, programming should be as efficient as possible in terms of memory allocation and time computations. Many improvements may be done in that direction, particularly we have not yet exploited the possibility of detecting and sharing equivalent expressions as we already do for S-words and S-letters. Furthermore, we also plan to analyze in terms of S-expressions, the convex, pointizable and Ord-Horn classes studied in the interval algebra theory [Nebel and Bürckert, 1995].

At the outside level, much work needs to be done to allow non-computer scientists to use the tool. Representing S-words graphically could be a first step. Next we could think of a tool for helping the user defining constraints between items resulting in a set of S-words graphically .

Acknowledgements

The authors would like to thank the referees for their constructive reports.

References

[Allen, 1983] Allen, J. F. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[Autebert and Schwer, 2003] Autebert, J.-M. and Schwer, S. R. On generalized Delannoy paths. *Journal on Discrete Mathematics*, 16(2):208–223, 2003.

[Campbell and Habermann, 1974] Campbell, R. H. and Habermann, A. N. The specification of process synchronization by path expressions. In *Lecture Notes in Computer Science*, volume 16, pages 89–102, 1974.

[David et al., 2005] David, M. D., Gabbay, D. M., and (eds), L. V. *Handbook of temporal reasoning in artificial intelligence*. Elsevier, 2005.

[Dubois and Schwer, 2000] Dubois, M. and Schwer, S. R. Classification topologique des ensembles convexes de Allen. In *R.F.I.A. 2000*, pages 59–68, 2000.

[Durand, 2008] Durand, I. A tool for handling s-languages `http://dept-info.labri.u-bordeaux.fr/~idurand/SLS`, 2008.

[Hamblin, 1969] Hamblin, C. L. Starting and stopping. *The Monist*, 53(3):410–425, 1969.

[Jonsson and Tarski, 1952] Jonsson, B. and Tarski, A. Boolean algebras with operators, ii. *American Journal of Mathematics*, 74:127–162, 1952.

[Krokhin et al., 2003] Krokhin, A., Jeavons, P., and Jonsson, P. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *Journal of the ACM*, 50:591–640, 2003.

[Ladkin and Maddux, 1994] Ladkin, P. and Maddux, R. On binary constraint problems. *Journal of the ACM*, 41 (3):435–469, 1994.

[Mancel et al., 2002] Mancel, C., Lopez, P., Rivière, N., and Valette, R. Relationships between Petri nets and constraint graphs: application to manufacturing. In *15th IFA world congress*, 2002.

[Manna and Pnueli, 1981] Manna, Z. and Pnueli, A. Verification of concurrent programs: Temporal proof principles. In Kozen, D., editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 200–252. Springer, 1981.

[Mörchen, 2006] Mörchen, F. A better tool than allen's relations for expressing temporal knowledge in interval data. In *Workshop on Temporal Data Mining at the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[Nebel and Bürckert, 1995] Nebel, B. and Bürckert, H. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[Nicod, 1923] Nicod, J. *La géométrie dans le monde sensible*. Alcan, Paris, 1923.

[Nökel, 1990] Nökel, K. Convex relations between time intervals. In *Proceedings of the 8th AAAI-90, vol III*, pages 721–727, 1990.

[Peterson, 1983] Peterson, J. *Petri Net Theory and the Modeling of Systems*. Pearson Educationr, 1983.

[Pratt, 1986] Pratt, V. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[Randell et al., 1992] Randell, D., Cohn, A., and Cui, Z. Computing transitivity tables: A challenge for automated theorem provers. In *Proceedings CADE, LNCS*, 1992.

[Reichenbach, 1947] Reichenbach, H. *Elements of Symbolic Logic*. Free Press, New-York, 1947.

[Revault, 1996] Revault, J. *Une modélisation par le graphe de la relation meet pour traiter des contraintes temporelles exprimées à l'aide d'intervalles.* Phd thesis, Université de Nantes, 1996.

[Rozenberg and Salomaa, 1996] Rozenberg, G. and Salomaa, A. *Handbook of Formal Languages: Word, Language, Grammar*, volume 58 of *Lecture Notes in Computer Science*. Springer, 1996.

[Russell, 1914] Russell, B. *Our knowledge of the External World*. G. Allen & Unwin,, London, 1914.

[Schwer, 2002] Schwer, S. R. S-arrangements avec répétitions. *Comptes Rendus de l'Académie des Sciences, Mathématiques*, 334:261–266, 2002.

[Schwer, 2004] Schwer, S. R. Relations temporelles qualitatives et langages formels. In *Actes des 11èmes Journées de Rochebrune, Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels, Le temps dans les systèmes complexes naturels et artificiels*, pages 329–340, 2004.

[Schwer, 2007] Schwer, S. R. Temporal reasoning without transitive tables, 2007. arXiv:0706.1290v1 [cs.AI].

[Strandh and Moore, 2002] Strandh, R. and Moore, T. A free implementation of CLIM. In *Proceedings of the International Lisp Conference*, San Francisco, California, 2002.

[Tarski, 1941] Tarski, A. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.

[van Beek, 1990] van Beek, P. Reasoning about qualitative temporal information. In *Proceedings of the 8th AAAI-90*, pages 728–734, 1990.

[van Beek and Cohen, 1990] van Beek, P. and Cohen, R. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–382, 1990.

[Vilain et al., 1989] Vilain, M., Kautz, H., and van Beek, P. Constraint propagation algorithms for temporal reasoning: a revisited report. In *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381, 1989.

[Whitehead, 1920] Whitehead, A. N. *The concept of nature*. Cambridge University Press, Cambridge, 1920.