Angelic Semantics for SiLCC

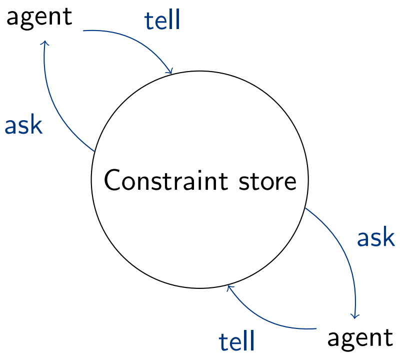
Thierry Martinez

Équipe-Projet Contraintes, INRIA Paris-Rocquencourt

Internal Seminar, 10 March 2010

Linear Concurrent Constraint Programming

- Concurrent constraint programming, V. Saraswat, ’93
- Linear concurrent constraint programming, [E. Best, F. S. de Boer, C. Palamidessi, ’97] [F. Fages, P. Ruet, S. Soliman, ’01]



$$
\begin{aligned}
A ::= &\mid C \\
&\mid \forall \vec{x}(C \rightarrow A) \\
&\mid \forall \vec{x}(C \Rightarrow A) \\
&\mid A \parallel A \\
&\mid \exists x(A)
\end{aligned}
$$

programs	=	formulas
executions	=	proof search

Operational semantics, logical observation of accessibility, modularity

tell $$\overline{(X; c; d, \Gamma) \rightarrow (X; c \otimes d; \Gamma)}$$

ask $$\frac{c \vdash d[\vec{x} := \vec{t}] \otimes e}{(X; c; \forall \vec{x}(d \rightarrow a), \Gamma) \rightarrow (X; e; a[\vec{x} := \vec{t}], \Gamma)}$$

persistent ask $$\frac{c \vdash d[\vec{x} := \vec{t}] \otimes e}{(X; c; \forall \vec{x}(d \Rightarrow a), \Gamma) \rightarrow (X; e; a[\vec{x} := \vec{t}], \forall \vec{x}(d \Rightarrow a), \Gamma)}$$

Theorem (Fages, Ruet, Soliman '01)

$$\mathcal{LL}^{store}(a) = \Downarrow\mathcal{O}_{\mathrm{a}}(a)$$

Modularity through variable hiding (Haemmerlé '08), EMOP...

Implementation(s) of LCC

SiLCC: an implementation of the LCC programming language.

- Bootstrap & modular definition of constraint systems (CHRat)
- Observation of ask firing
  - Asks equipped with side-effects
  - Asks firing $\rightsquigarrow \Downarrow\mathcal{O}_{\mathrm{a}}(a)$
- Committed-choice semantics: compilation from LCC to CHR
  - Monolithic guards: bad modularity,
  - Not an ideal framework to bootstrap,
  - The programmer should ensure that the scheduler can *only* make the good choice
- *Angelic* semantics
  - Modularity for checking guard entailment: decomposition of guards
  - Sound with respect to logical semantics
  - In terms of observability, the scheluder always makes the good choice!
  - How to compute the set of logical consequences for an agent?

CHRat modularization of constraint system

$$H \iff G \mid B$$

is translated into

$$\begin{array}{rcl} H & \Longrightarrow & \mathrm{ask}(G) \\ H, \mathrm{entailed}(G) & \iff & B \end{array}$$

- H should only be consumed if G is entailed.
- This translation relies on refined semantics for the generated code,
- but only guarantees that naive semantics of the source code is preserved:
  - propagations can be fired more than once,
  - there is no longer control on the order of rule firing
- Therefore, CHRat is not built on a CHRat kernel:
  not ideal to bootstrap!

CHR propagation *v.s.* angelic scheduling

- Propagation is a fundamental construction in CHR to circumvent committed-choice: allows to trigger computation without consumption,
- but propagation is not captured by any logical reading

- *An angelic scheduler always makes the good choice*: in terms of observability, allows to trigger computation only in the case that making the consumption *is* the good choice.
- Sound with logical semantics

Guard decomposition: the simple case

Lemma

For all agent A and constraints c_1 and c_2,

$$\Downarrow\mathcal{O}_{\mathrm{a}}(A \parallel (c_1 \otimes c_2 \Rightarrow a)) = \Downarrow\mathcal{O}_{\mathrm{a}}(A \parallel (c_1 \Rightarrow c_2 \rightarrow a))$$

(Angelic equivalence)

This makes room to trigger computation to check c_2:

$$c_1 \Rightarrow \exists k(\mathrm{check\_entailment}(c_2, k) \parallel (\mathrm{true}(k) \rightarrow a))$$

Guard decomposition: the general case

With projections on observables removing $\mathrm{t}(k)$ control tokens, the agent

$$\forall \vec{x}_1 \dots \vec{x}_n (c_1 \otimes \cdots \otimes c_n \rightarrow a)$$

with $\vec{x}_i \cap \mathrm{fv}(c_1 \otimes \cdots \otimes c_{i-1}) = \emptyset$ is angelically equivalent to

$$\exists k, \mathrm{t}(k) \parallel \forall \vec{x}_1 (c_1 \Rightarrow \cdots \Rightarrow \forall \vec{x}_n (c_n \Rightarrow \mathrm{t}(k) \Rightarrow a) \dots)$$

Mixing this result with the previous lemma, the agent

$$\forall \vec{x}_1 \vec{x}_2 \dots \vec{x}_n (c_1 \otimes c_2 \otimes \cdots \otimes c_n \Rightarrow a)$$

is angelically equivalent to

$$\forall \vec{x}_1 (c_1 \Rightarrow \exists k, \mathrm{t}(k) \parallel \forall \vec{x}_2 (c_2 \Rightarrow \cdots \Rightarrow \forall \vec{x}_n (c_n \Rightarrow \mathrm{t}(k) \Rightarrow a) \dots))$$

Consequence: for the kernel, persistent asks on atomic constraints suffice.

Thierry.Martinez@inria.fr

A minimal constraint system

Hypotheses

- an infinite set of variables $\mathcal{V}$;
- an infinite domain of values (including $\mathbb{N}$);
- a signature Σ for linear predicates.

$$
\begin{aligned}
C ::= & \mid 1 \\
& \mid p(\vec{v}) \\
& \mid C \otimes C \\
& \mid \exists x(C)
\end{aligned}
$$

Rules of ILL, decidable entailment (in $\mathbf{O}(n^2)$!)

Example: Scalar product computation

Suppose two built-in agents for arithmetic calculation behaving as:

- $\forall x_1 x_2 k(\text{product}(x_1, x_2, k) \Rightarrow \text{value}(k, x_1 \times x_2))$
- $\forall x_1 x_2 k(\text{sum}(x_1, x_2, k) \Rightarrow \text{value}(k, x_1 + x_2))$

(Needed because the kernel is not supposed to contain arithmetic anymore, to be compared with the *is* primitive of Prolog.)

Agent for the scalar product

$$\begin{aligned}\forall x_1 x_2 y_1 y_2 p(\text{scalar}(x_1, y_1, x_2, y_2, p) \Rightarrow \\ \exists k_x \exists k_y (&\text{product}(x_1, x_2, k_x) \parallel \\ &\text{product}(y_1, y_2, k_y) \parallel \\ &\forall p_x p_y (\text{value}(k_x, p_x) \otimes \text{value}(k_y, p_y) \rightarrow \\ &\quad \text{sum}(p_x, p_y, p))))\end{aligned}$$

Two derivation paths for scalar product computation

$$(\emptyset; \mathrm{scalar}(1,2,3,4,p); \Gamma)$$
$$\downarrow$$
$$(V; \mathrm{p}(1,2,k_x) \otimes \mathrm{p}(3,4,k_y); \Gamma, A)$$

$(V; \mathrm{v}(k_x,2) \otimes \mathrm{p}(3,4,k_y); \Gamma, A)$		$(V; \mathrm{p}(1,2,k_x) \otimes \mathrm{v}(k_y,12); \Gamma, A)$
$\downarrow$		$\downarrow$
$(V; \mathrm{v}(k_x,2) \otimes \mathrm{v}(k_y,12); \Gamma, A)$		$(V; \mathrm{v}(k_x,2) \otimes \mathrm{v}(k_y,12); \Gamma, A)$
$\downarrow$	Scheduling non-determinism	$\downarrow$
$(V; \mathrm{s}(2,12,p); \Gamma)$		$(V; \mathrm{s}(2,12,p); \Gamma)$
$\downarrow$		$\downarrow$
$(V; \mathrm{v}(p,14); \Gamma)$		$(V; \mathrm{v}(p,14); \Gamma)$

Notations

$$\Gamma = \{\!| \forall x_1 x_2 k(\mathrm{p}(x_1,x_2,k) \Rightarrow \mathrm{v}(k, x_1 + x_2)), \forall x_1 x_2 k(\mathrm{s}(x_1,x_2,k) \Rightarrow \mathrm{v}(k, x_1 \times x_2)) \\ \forall x_1 x_2 y_1 y_2 p(\mathrm{scalar}(x_1,y_1,x_2,y_2,p) \Rightarrow \exists k_x \exists k_y (\mathrm{p}(x_1,x_2,k_x) \| \mathrm{p}(y_1,y_2,k_y) \| A)) |\!\}$$

$$A = \langle \forall p_x p_y (\mathrm{v}(k_x,p_x) \otimes \mathrm{v}(k_y,p_y) \multimap \mathrm{s}(p_x,p_y,p)) \rangle, V = \{k_x, k_y\}$$

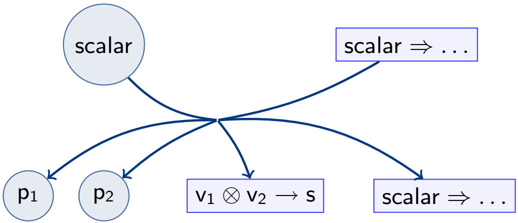
Derivation net: informal definition

A derivation net is a (potentially infinite) labeled and oriented multihypergraph where

- each vertex is labeled with either a linear predicates or an ask,
- hyperedges stand for firings: for each edge,

  sources exactly one ask plus matching linear predicates,

  targets asks and linear predicates appearing in the ask body.



No unicity for derivation nets: a derivation net is a representation for a strategy of angelic execution.

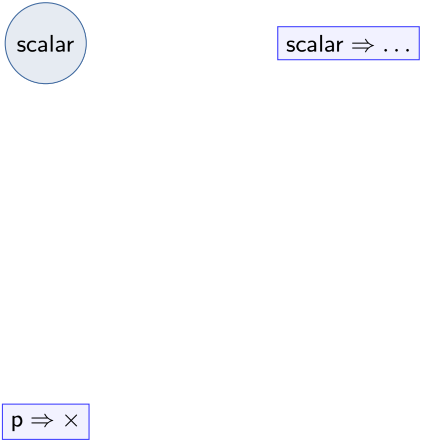
Derivation net

Definition (Derivation net)

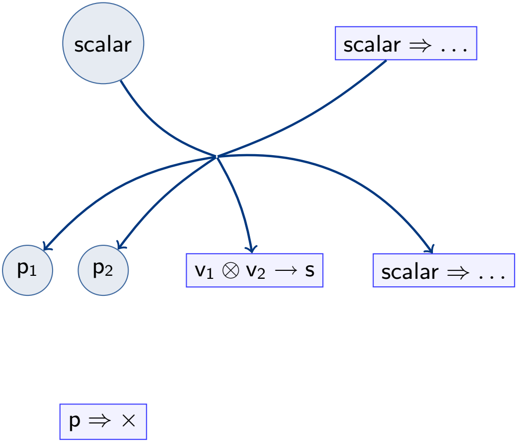
A derivation net is a (potentially infinite) oriented multihypergraph (V, E, i) with a vertex labeling $\ell : V \to \mathcal{T} \cup \mathcal{A}$ where

- vertices V are labeled with
  - linear predicates $\mathcal{T} = \{p(v_1, \dots, v_n) \mid p/n \in \Sigma \text{ and } v_1, \dots v_n \in \mathcal{V}\}$
  - asks $\mathcal{A} = \{\forall \vec{x}(c \to b), \forall \vec{x}(c \Rightarrow b) \mid \vec{x} \in \mathcal{V}^*, c \in C, b \in A\}$,
- for each edge $e \in E$, the three following conditions hold:
  1. $\ell({}^\bullet e) = \{\!|a, t_1, \dots, t_n|\!\}$ with $a \in \mathcal{A}$ and $t_1, \dots, t_n \in \mathcal{T}$
     let $a = \forall \vec{x}(c \to / \Rightarrow b)$
     and let $b \equiv \exists \vec{y}(t'_1 \parallel \cdots \parallel t'_m \parallel a_1 \parallel \cdots \parallel a_p)$ with $t_i \in \mathcal{T}$ and $a_j \in \mathcal{A}$,
  2. $t_1 \otimes \cdots \otimes t_n \vdash \sigma(c)$ with $\exists \vec{t}, \sigma = [\vec{x} := \vec{t}]$
  3. $\ell(e^\bullet) = \sigma\sigma'(\{\!|t_1, \dots, t_m, a_1, \dots, a_p|\!\})$ with $\exists \vec{t}', \sigma' = [\vec{y} := \vec{t}']$

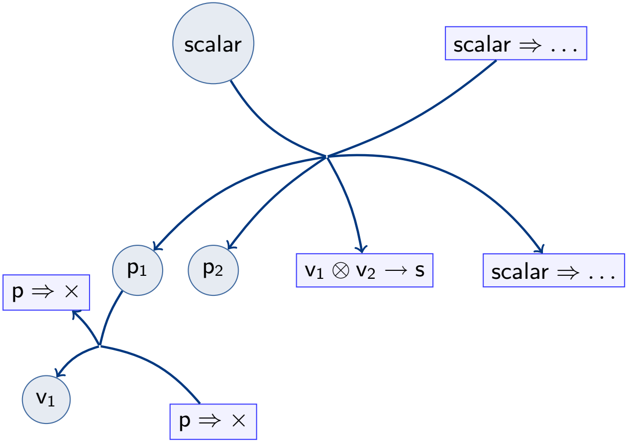
Derivation net for the scalar product



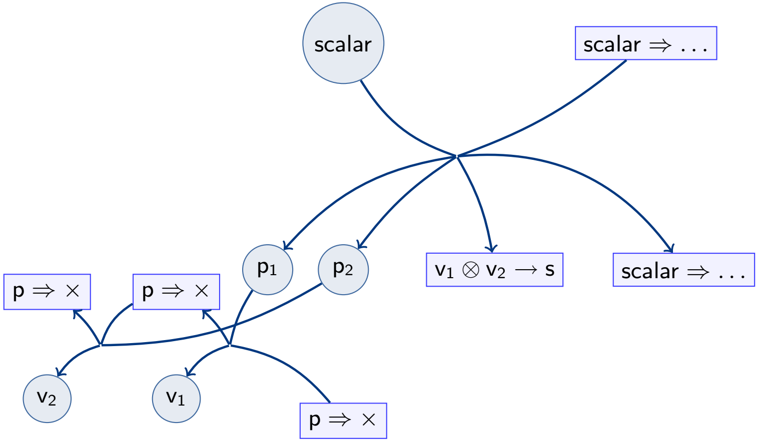
Derivation net for the scalar product



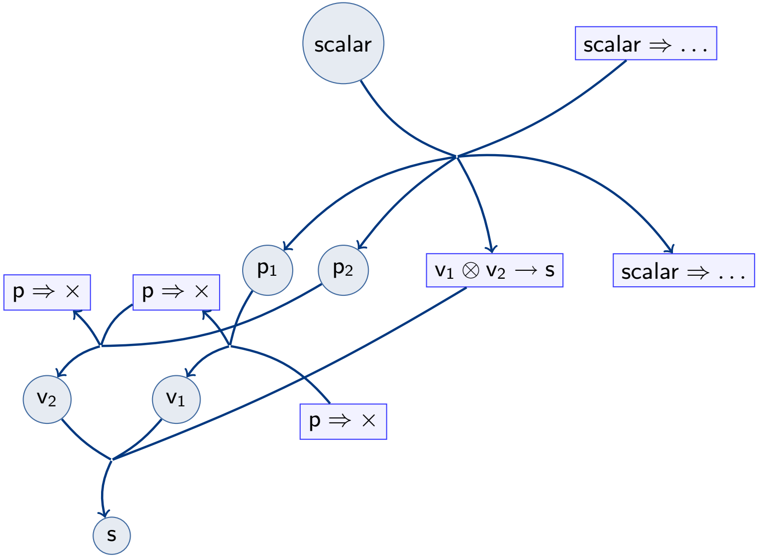
Derivation net for the scalar product



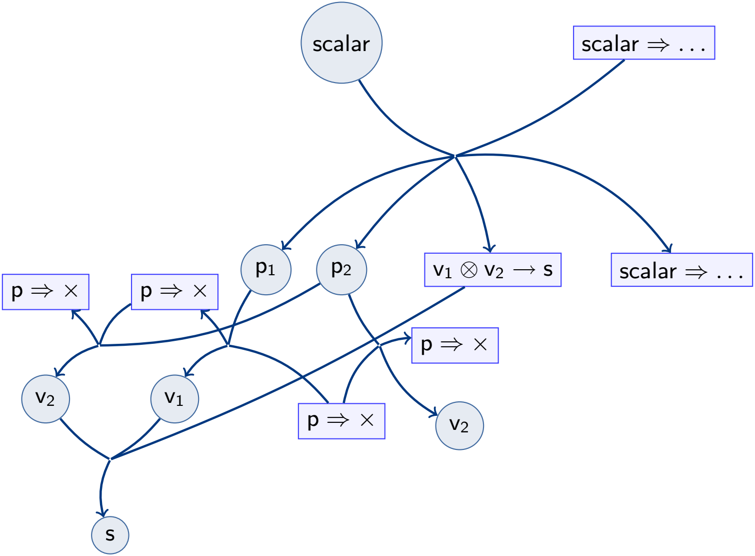
Derivation net for the scalar product



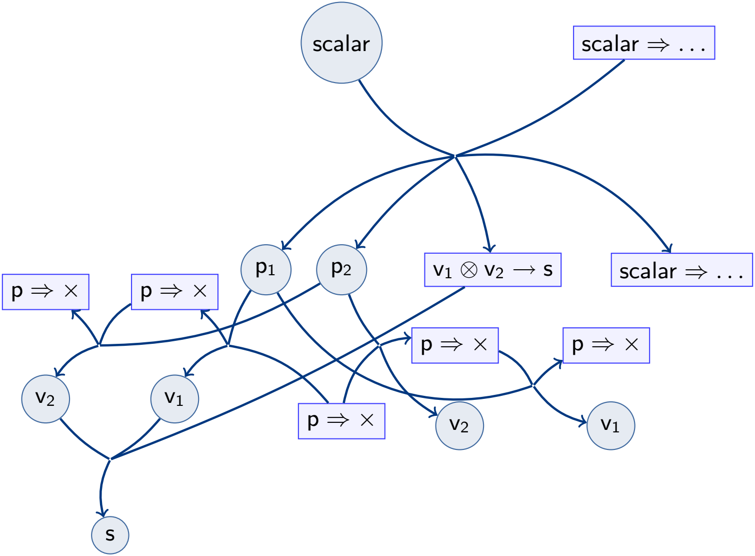
Derivation net for the scalar product



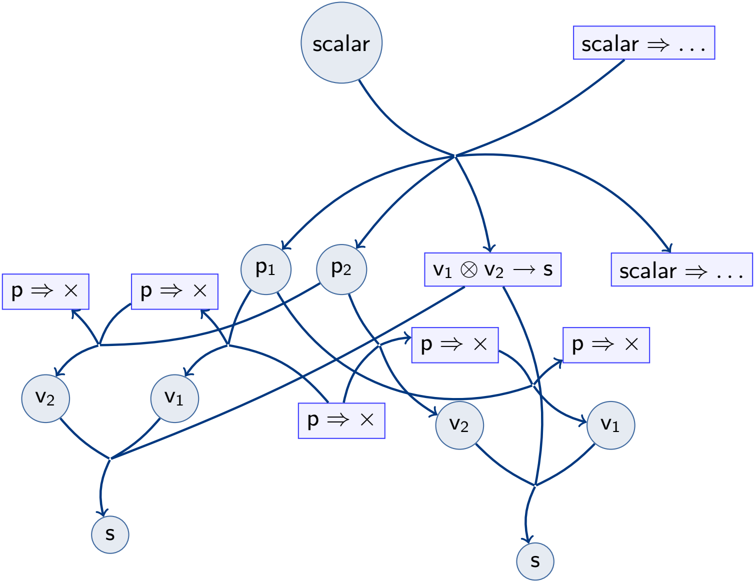
Derivation net for the scalar product



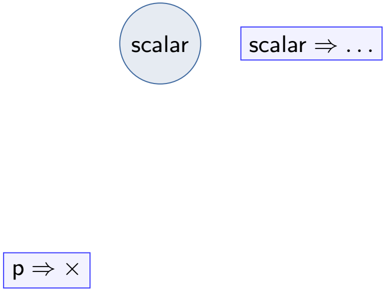
Derivation net for the scalar product



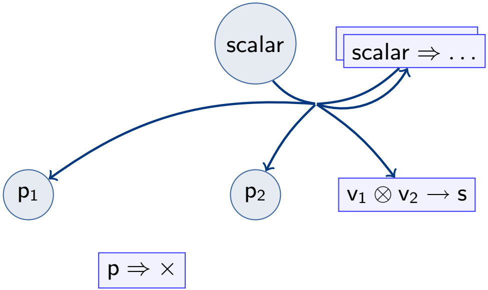
Derivation net for the scalar product



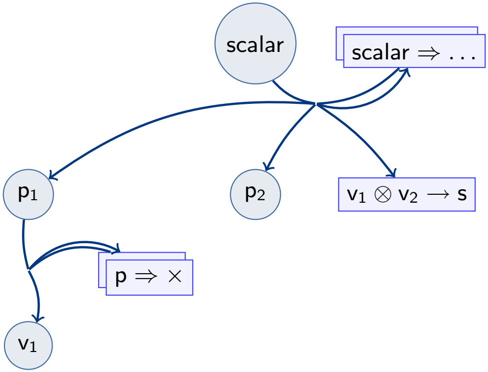
Derivation net for the scalar product with sharing of asks



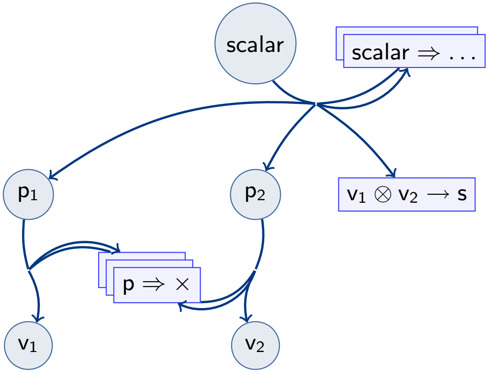
Derivation net for the scalar product with sharing of asks



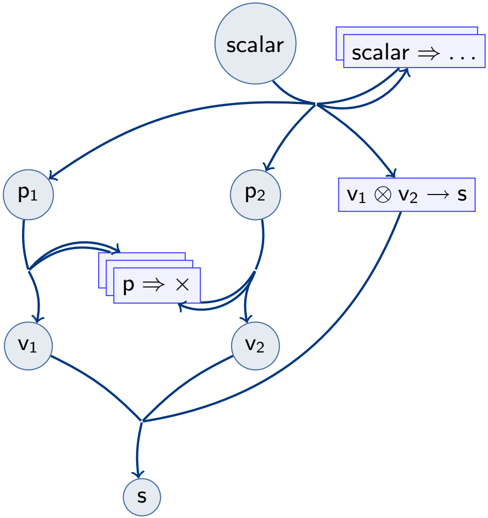
Derivation net for the scalar product with sharing of asks



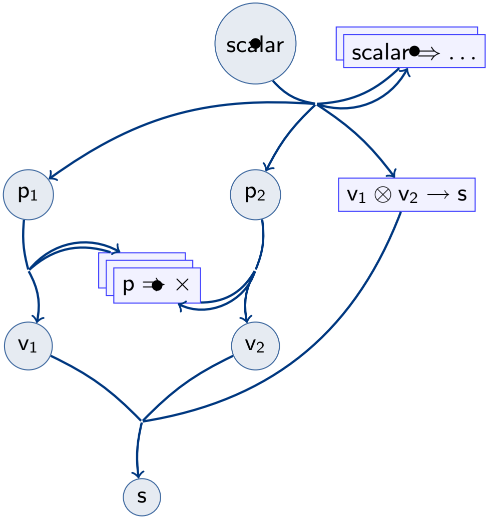
Derivation net for the scalar product with sharing of asks



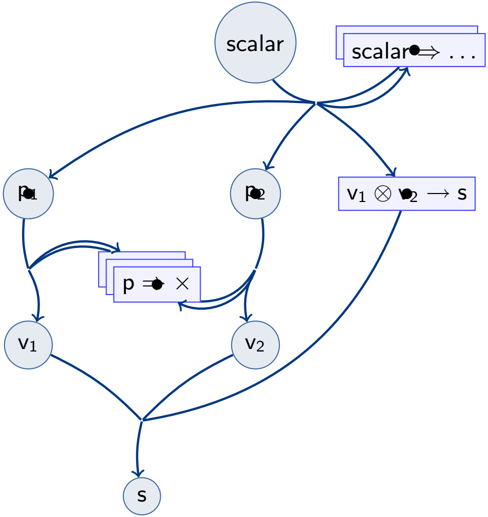
Derivation net for the scalar product with sharing of asks



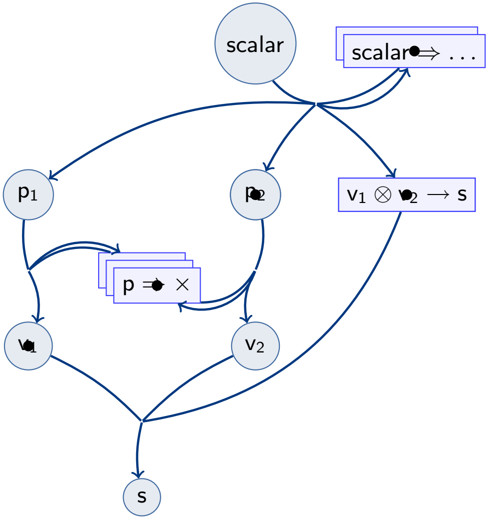
Petri-net interpretation



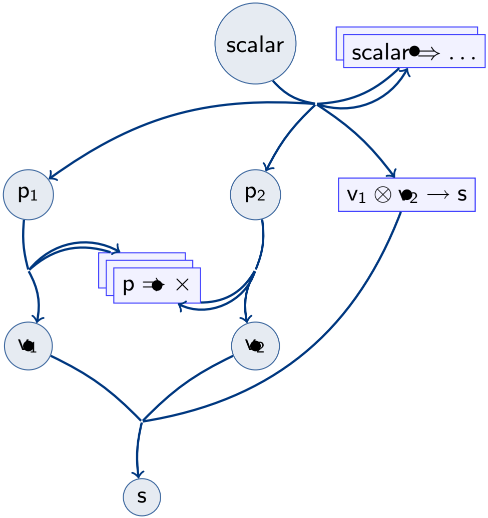
Petri-net interpretation



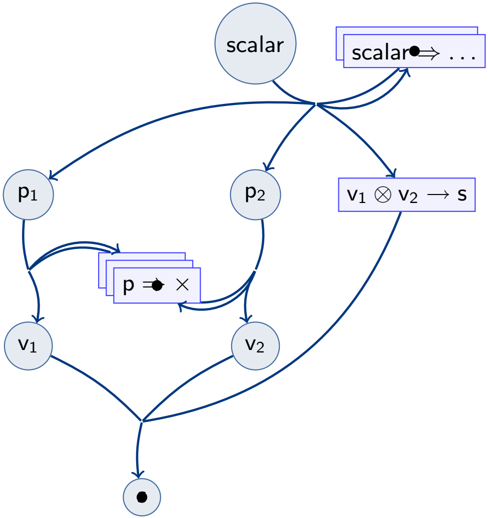
Petri-net interpretation



Petri-net interpretation



Petri-net interpretation



States & Derivations

Definition

A state s is a multiset of vertices: $V \to \mathbb{N}$.

Definition

There is a derivation $s \to_d s'$ when there exists an edge e such that

$$s' = s - {}^\bullet e + e^\bullet$$

(that is to say, $s' : v \mapsto s(v) - {}^\bullet e(v) + e^\bullet(v)$).

Definition

The set of accessible states from an initial state s forms the following observable:

$$\mathcal{O}_\mathrm{a}(d, s) = \{s' : V \to \mathbb{N} \mid s \xrightarrow{*}_d s'\}$$

Thierry.Martinez@inria.fr

Relating derivation nets with LCC operational semantics

For all agent $a \equiv \exists \vec{x}(a_1 \parallel \cdots \parallel a_m)$ with $a_i \in \mathcal{T} \cup \mathcal{A}$,

$$S(a) = \{s : V \to \mathbb{N} \mid \ell(s) = \{\!|a_1, \ldots, a_m|\!\}\}$$

Let $\mathcal{O}_\mathrm{a}(d, a) = \bigcup_{s \in S(a)} \mathcal{O}_\mathrm{a}(d, s)$.

Theorem (Correction)

For all agent a and for all derivation net d,

$$\mathcal{O}_\mathrm{a}(d, a) \subseteq \mathcal{O}_\mathrm{a}(a)$$

Theorem (Completeness)

For all agent a, there exists a complete derivation net d, such that

$$\mathcal{O}_\mathrm{a}(d, a) = \mathcal{O}_\mathrm{a}(a)$$

Iterative computation of a complete derivation net

Starting from initial the vertices of the initial state, does a breadth-first search among accessible edges.

Testing if an edge is accessible: decidable for all sharing strategy (Petri-net reachability).
Can be intractable in practice.

With the “sharing asks” strategy:

- Optimal non-determinism quotient under the hypothesis that the external observer has the ability to distinguish every tell (add to each token a unused argument carrying a hidden variable)
- Accessibility check in $\mathbf{O}(n \log n)$ in worst case, nearly always constant-time in practice.

Accessibility check with ask-sharing

Definition

The ancestor multihypergraph $\uparrow e$ of an edge e is the submultihypergraph with only vertices and edges such that there exists a path from them to e.

Definition

A conflict is a vertex with two successor edges.

Definition

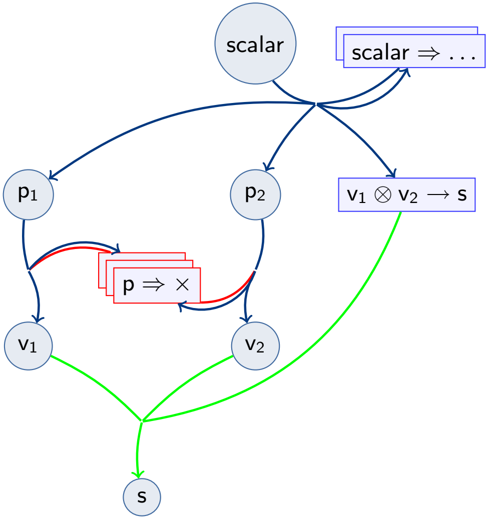
A multihypergraph is 1-bounded if from any 1-bounded state, there are only 1-bounded accessible states.

Derivation nets for ask sharing strategy are 1-bounded and have no non-trivial cycles.

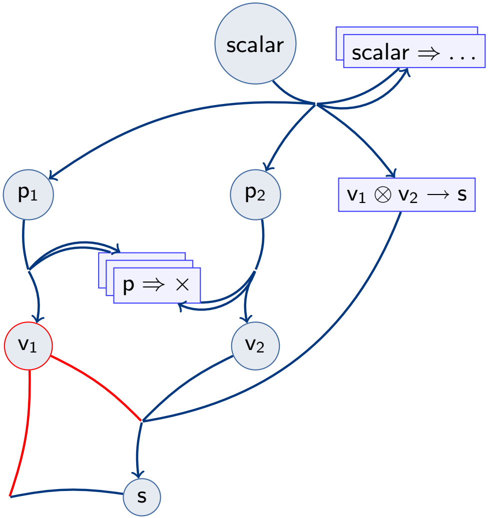
Lemma

In a 1-bounded multihypergraph without non-trivial cycle, an edge e is accessible if and only if all conflicts in $\uparrow e$ are in trivial cycles.

Safe Conflict



Unsafe Conflict



Accessibility algorithm

- Preparation: at each new vertex creation v_0, computes a table $t(v_0)$ which associates each ancestor vertice v (outside trivial cycles) to its potential immediate successor edge e (there is at most one!):

$$t(v_0) : v \mapsto e$$

With balanced binary trees, logarithmic time cost for each vertex.

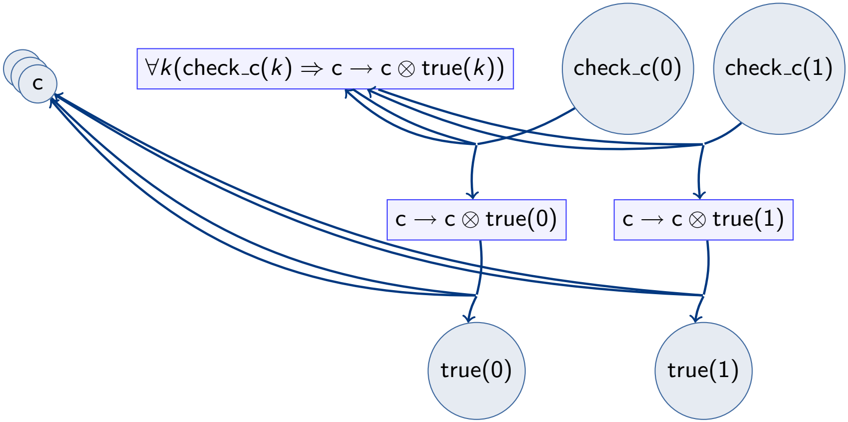
- To check if a binary edge e_0 between v_0 and v_1 introduces a conflict:
  1. choose one of the predecessor vertex, say v_0, (preferably the one with least ancestors)
  2. let $t \leftarrow t(v_1) + (v_1 \mapsto e_0)$
  3. begin with $v \leftarrow v_0$,
  4. for each predecessor vertex v' of each predecessor edge e of v,
  5. if $t(v')$ is defined, succeeds if $t(v') = e$ or v' in trivial cycle, else fails,
  6. if not, let $t \leftarrow t + (v' \mapsto e)$ and recursively go to 4 for $v \leftarrow v'$.

In worst case, logarithmic cost (table search) for each ancestor.

In practice, either edges between neighbours ($t(v')$ is often defined) or edges between a vertex and a top-level ask (with few ancestors).

Token sharing on immediate restoration

Typical case in classical constraint checking:



Trivial cycles for c. But not robust with guard decomposition...

Conclusion

- Angelic semantics:
  - modular decomposition of guards,
  - consistent with logical semantics
- Derivation nets:
  - flexible formalism to express scheduling non-determinism elimination through vertex sharing,
  - all sharing are decidable (Petri-net accessibility) even the optimal one (where all equal vertices are shared)
- Sharing of trivial cycles:
  - eliminates ask-*v.s.*-ask scheduling non-determinism,
  - optimal with the oracle which distinguishes every token,
  - execution preserves theoretical complexity class (each execution step is in polynomial time) and preserves complexity in practice (most execution steps are in constant time)
- Generalisation for some non-trivial cycles?
- Control for chaotic iteration of propagators?
- Garbage collection of some parts of the hypergraph?

Decidable entailment

Constraint normal form

$$c \equiv \exists x_1 \ldots \exists x_i (p_1(\vec{u}_1) \otimes \cdots \otimes p_m(\vec{u}_m))$$

Entailment criterion

Let

$$c \equiv \exists x_1 \ldots \exists x_i (p_1(\vec{u}_1) \otimes \cdots \otimes p_m(\vec{u}_m))$$
$$d \equiv \exists y_1 \ldots \exists y_j (q_1(\vec{v}_1) \otimes \cdots \otimes q_n(\vec{v}_n))$$

$c \vdash_{\mathcal{C}} d$ is equivalent to
$\{\!|p_1(\vec{u}_1)\rho, \ldots, p_m(\vec{u}_m)\rho|\!\} = \{\!|q_1(\vec{v}_1)\sigma, \ldots, q_n(\vec{v}_n)\sigma|\!\}$ where

- ρ is a renaming which maps $\{x_1, \ldots, x_i\}$ to fresh variables with respect to d.
- σ is a substitution supported by $\{y_1, \ldots, y_j\}$.

Oriented multigraphs

An oriented multigraph is given as a tuple (V, i) where

- V is a set of vertices,
- $i : V \times V \to \mathbb{N}$ is an incidence function.

Prevertices and postvertices

For all $v \in V$, let ${}^\bullet v$ be the multiset of prevertices and $v^\bullet$ be the multiset of postvertices defined as follows

$${}^\bullet v : u \mapsto i(u, v), v^\bullet : u \mapsto i(v, u)$$

Hypermultigraphs

An oriented multigraph is bipartite when there exists a partition $V = V_0 \uplus V_1$ such that for all $v, v' \in V$, if $\overline{v} = \overline{v'}$, then $i(v, v') = 0$.

An oriented hypermultigraph is given as a tuple (V, E, i) where $(V \uplus E, i)$ is a bipartite oriented multigraph.

Homomorphic images of multisets

Let V and S be two sets and $f : V \rightarrow$ s. (For example, V is a set of vertices and S a set of labels).

For all multiset $m : V \rightarrow \mathbb{N}$, let $f(m)$ be the multiset

$$f(m) : x \in S \mapsto \sum_{\substack{v \in V \\ f(v)=x}} m(v)$$