Temporal Logics for Concurrent Recursive Programs: Satisfiability and Model Checking

Benedikt Bollig, Aiswarya. Cyriac, Paul Gastin, Marc Zeitoun

LSV, ENS Cachan — LaBRI, U. Bordeaux — CNRS — INRIA

MFCS 2011

Motivation

- Verification of concurrent recursive systems against temporal logics.
- We make the verification generic wrt.
  - the models: capture Words, Trees, Nested Words, Mazurkiewicz traces

Motivation

- Verification of concurrent recursive systems against temporal logics.
- We make the verification generic wrt.
  - the models: capture Words, Trees, Nested Words, Mazurkiewicz traces
  - the temporal logics: capture LTL, CTL, PDL (with $^{-1}$and $\cap$), XPath.

Motivation

- Verification of concurrent recursive systems against temporal logics.
- We make the verification generic wrt.
  - the models: capture Words, Trees, Nested Words, Mazurkiewicz traces
  - the temporal logics: capture LTL, CTL, PDL (with $^{-1}$and $\cap$), XPath.
  - the decidabillity proofs: extend results to both concurrent and recursive
  - the complexity bounds for verifying such concurrent recursive systems.

Main Tools

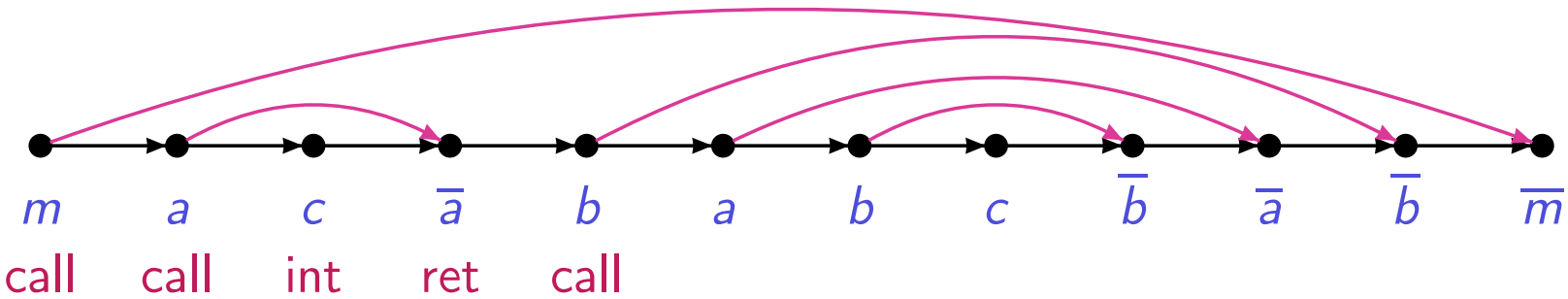
1. Linearizations of concurrent recursive behaviors: multiply nested words.
2. Tree encoding of multiply nested words.
3. Adaptation of existing results for trees.

Recursive Systems

```
main():
    call a()
    call b()
    return
```

```
a():
    call b() or do c
    return
```

```
b():
    call a() or do c
    return
```

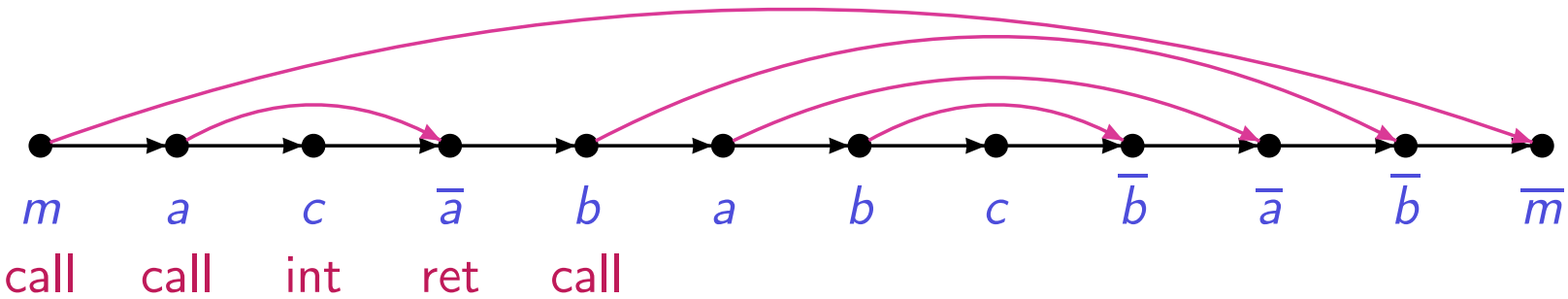


Recursive Systems

```
main():
    call a()
    call b()
    return
```

```
a():
    call b() or do c
    return
```

```
b():
    call a() or do c
    return
```



Nested Words [Alur, Madhusudan 04–] : $(V, \text{act}, \text{type}, \text{succ}, cr)$

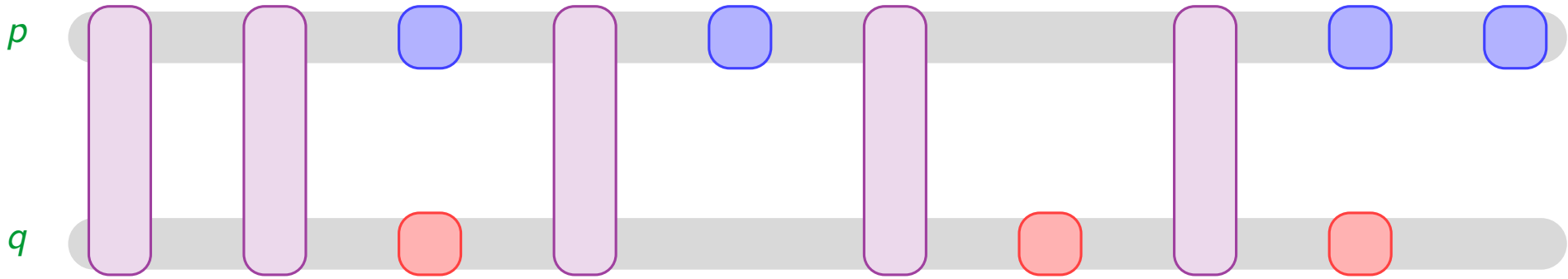
- V: set of nodes (events).
- $\text{act} : V \to \text{Labels}$
- $\text{type} : V \to \{\text{call}, \text{ret}, \text{int}\}$.
- $\text{succ} \subseteq V \times V$ is the linear successor.
- $cr \subseteq V \times V$ is the matching relation.

here, $\text{Labels} = \{a, b, c, m, r\}$.

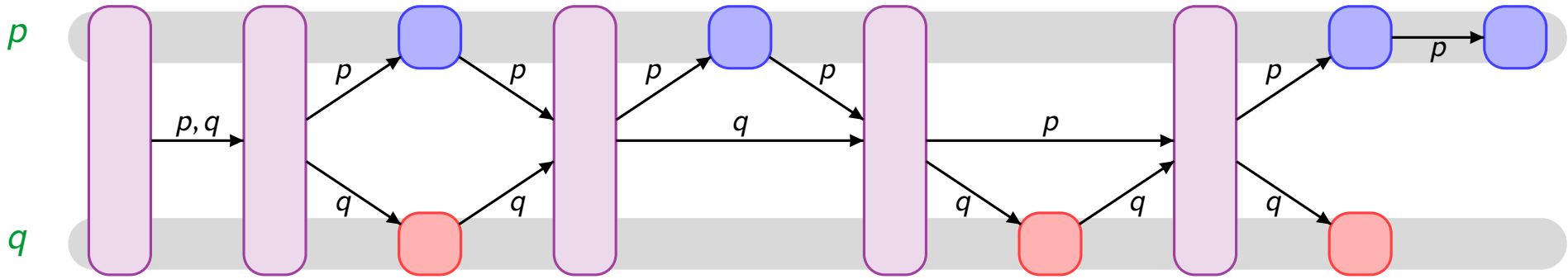
calls/returns may be unmatched

Concurrent Systems

Concurrent Systems

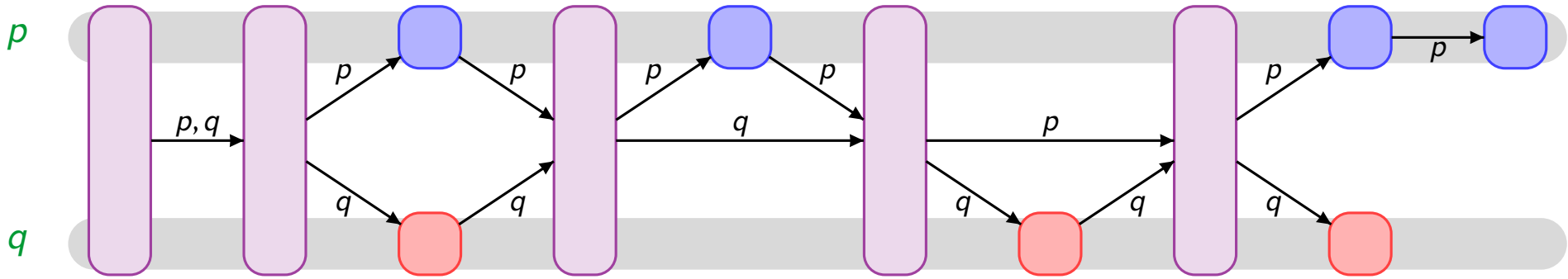


Concurrent Systems

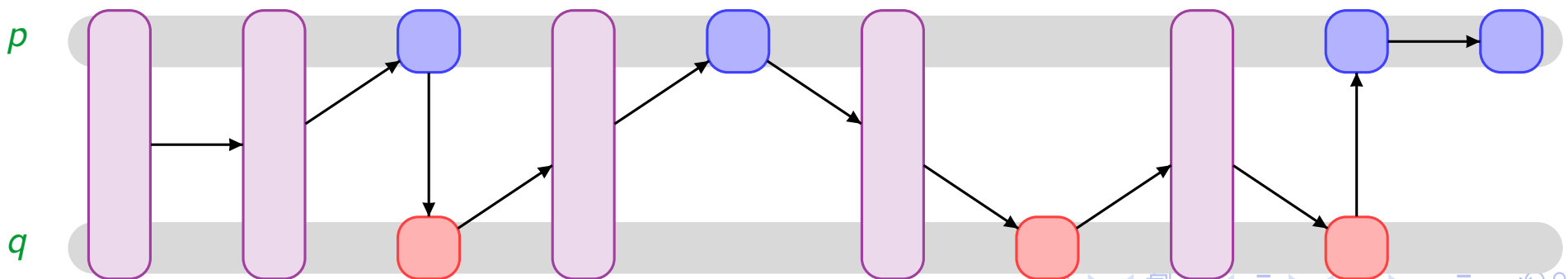


- Mazurkiewicz Trace: $(V, \text{act}, \text{proc}, <_p)$
- $proc : V \to \{p, q\}$, and $<_p$ order on process p. Yield partial ordering:
  - Two nodes sharing a process must be ordered.
  - Two consecutive nodes must share a process.

Concurrent Systems

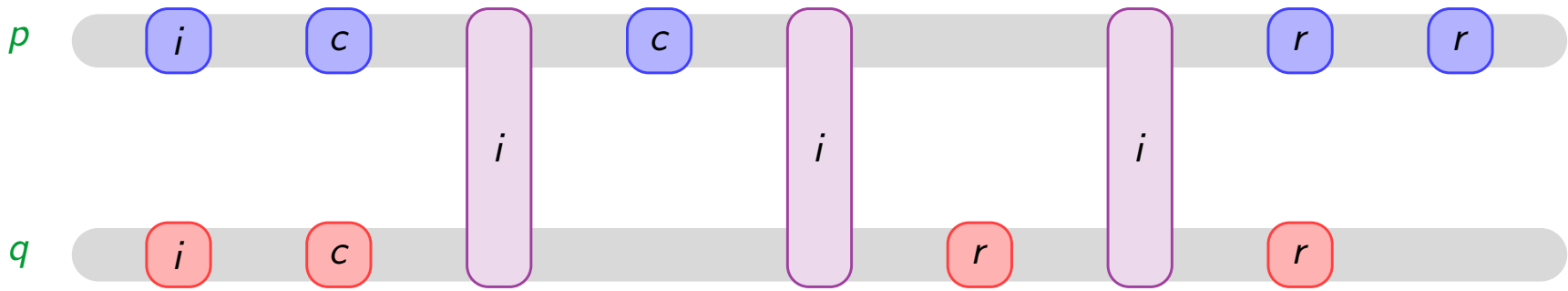


- Mazurkiewicz Trace: $(V, \text{act}, \text{proc}, <_p)$
- $proc : V \to \{p, q\}$, and $<_p$ order on process p. Yield partial ordering:
  - Two nodes sharing a process must be ordered.
  - Two consecutive nodes must share a process.
- We use linearizations of such traces:

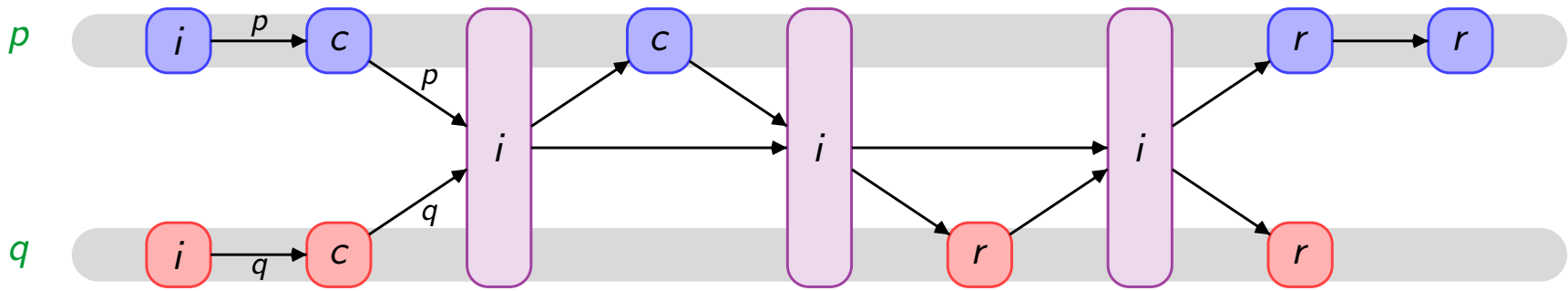


Concurrent Recursive Systems

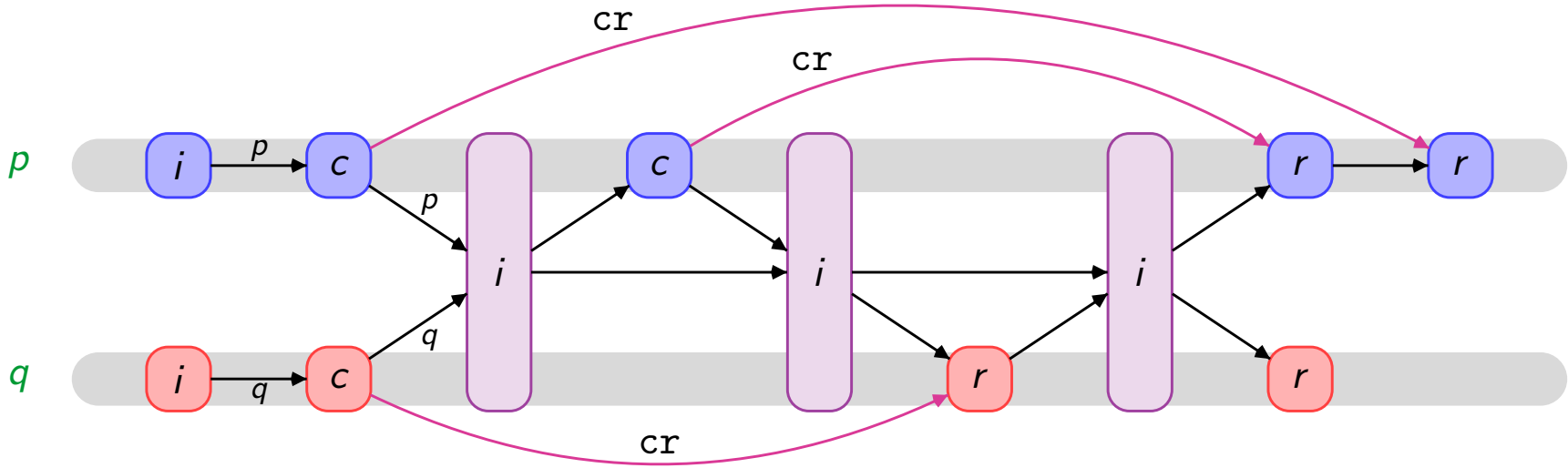
Concurrent Recursive Systems



Concurrent Recursive Systems

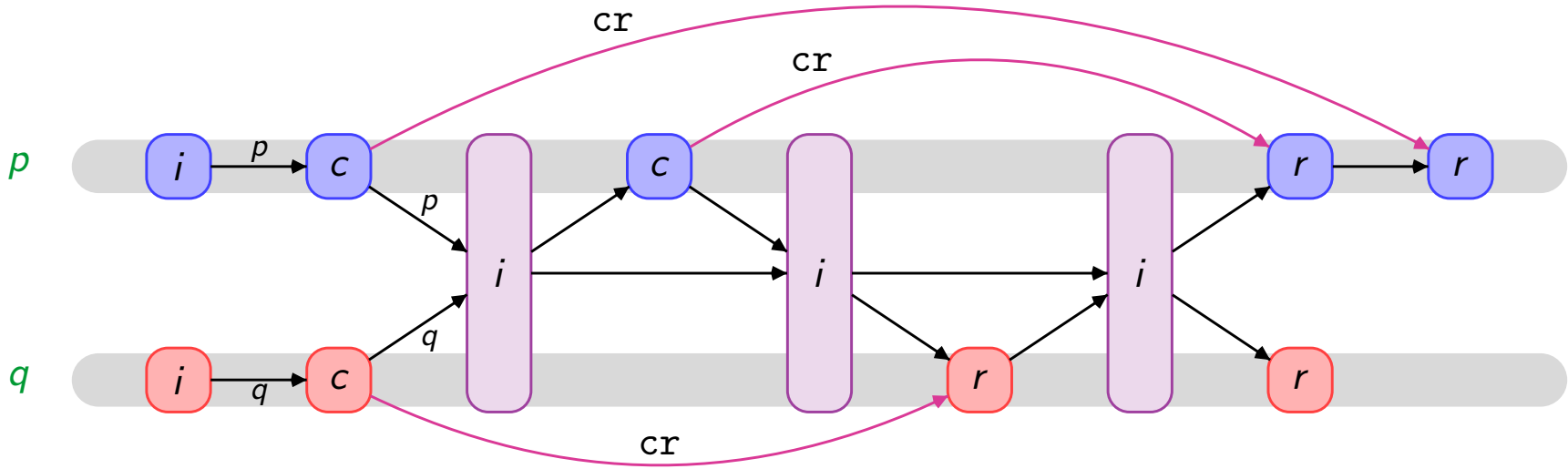


Concurrent Recursive Systems

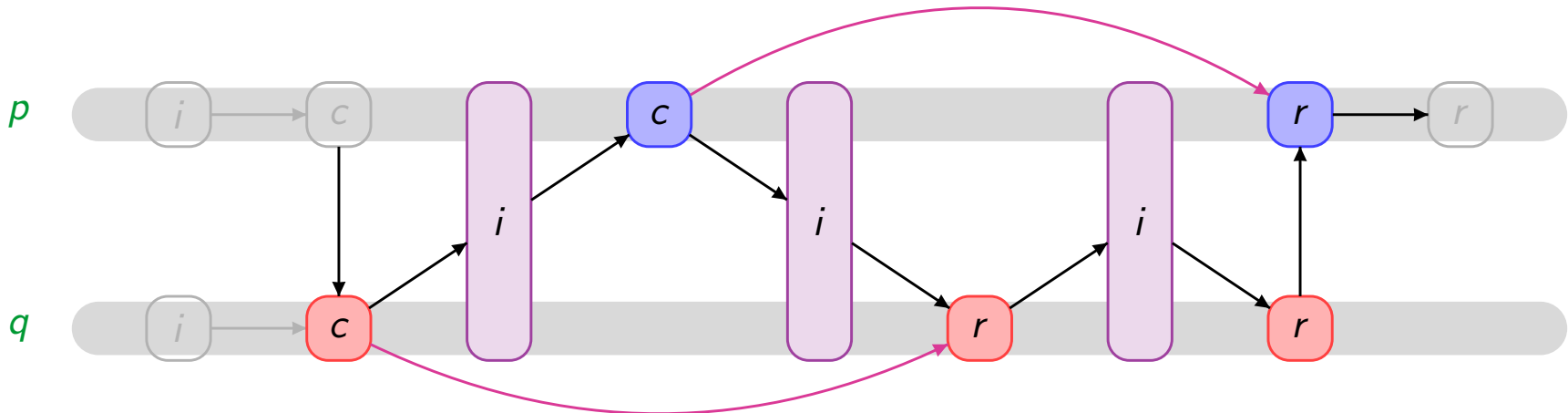


- Nested traces: $(V, \mathsf{act}, \mathsf{proc}, \mathsf{type}, <_p, <_q, \mathsf{cr})$.
- Calls and returns local to one process.

Concurrent Recursive Systems

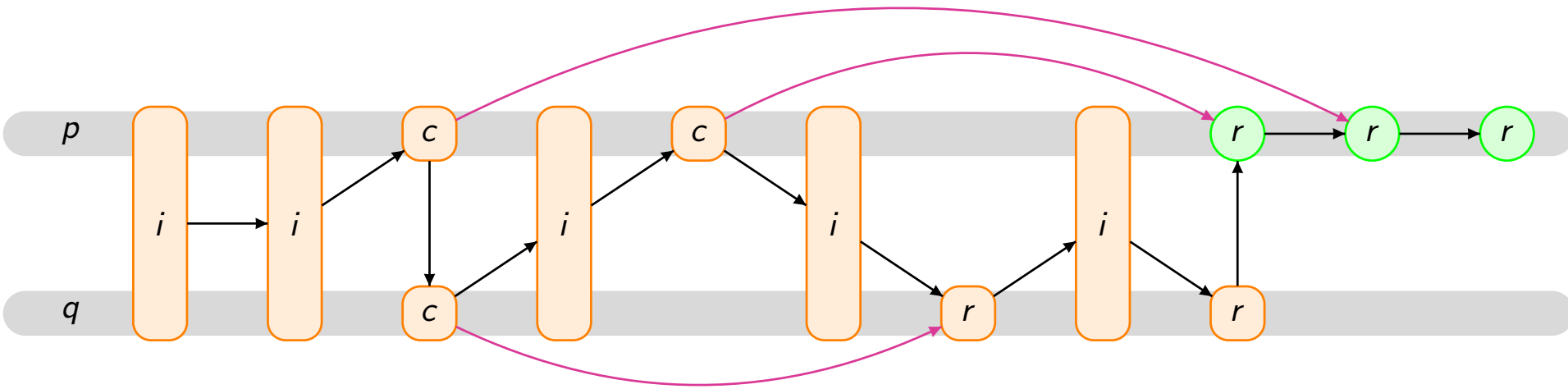


- Nested traces: $(V, \mathsf{act}, \mathsf{proc}, \mathsf{type}, <_p, <_q, \mathsf{cr})$.
- Calls and returns local to one process.
- ⚠ Linearizations need not be nested words!



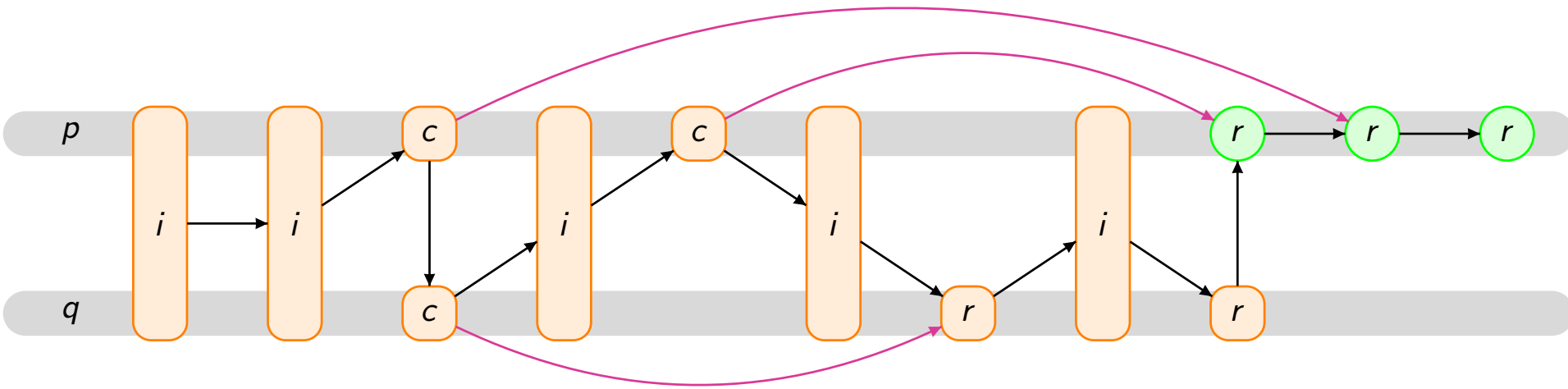
Phases

- We limit ourselves to k-phase nested traces.
- A trace is k-phase if
  - it has a linearization that can be split into k factors.
  - in each factor, only one single process can return.



Phases

- We limit ourselves to k-phase nested traces.
- A trace is k-phase if
  - it has a linearization that can be split into k factors.
  - in each factor, only one single process can return.



- For traces with unbounded phase, crossings of call-return edges of two stacks $\rightsquigarrow$ encoding of the grid.

A generic model: Labeled structures

- Examples models: Words, Trees, Mazurkiewicz traces, Nested traces.
- Models = labeled graphs.
- Signature $\mathcal{S} = (\Sigma, \Gamma)$. node labels Σ, edge labels Γ.
- $\mathcal{S}$-graph $G = (V, \lambda, \nu)$
- Node-labeling $\lambda : V \to 2^{\Sigma}$, and Edge-labeling $\nu : (V \times V) \to 2^{\Gamma}$.
  Graph induced $u \to v$ iff $\nu(u, v) \neq \emptyset$ well-founded and acyclic.

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

$$M = \mathsf{X}(\alpha),\ \mathsf{U}(\alpha, \beta)$$
$$[\![\mathsf{X}]\!](x, Y) = \exists y\,(y \in Y \wedge \mathrm{succ}(x, y))$$
$$[\![\mathsf{U}]\!](x, Y, Z) = \exists z\,(x \leq z \wedge z \in Z \wedge \forall y(x \leq y < z) \Rightarrow y \in Y)$$

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

  EGα = $\exists$ infinite path from the current node where α always holds.

$$[\![\mathrm{EG}]\!](x, X) = \exists Y\,(Y \subseteq X \wedge x \in Y \wedge \forall z\,(z \in Y \rightarrow \exists z'\,(z' \in Y \wedge z \text{ succ } z')))$$

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

A generic temporal logic over $\mathcal{S}$-graphs

Node formulas $\varphi ::= \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$

Path expressions $\pi ::= ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$

- M is an MSO($\mathcal{S}$) modality.
- Captures LTL, CTL, PDL (with $\cap$ and converse), regular XPath, NWTL

$$M = \mathsf{X}(\alpha), \mathsf{Y}(\alpha), \mathsf{X}^{\mathtt{cr}}(\alpha), \mathsf{Y}^{\mathtt{cr}}(\alpha), \mathsf{U}^{\mathtt{s}}(\alpha,\beta), \mathsf{S}^{\mathtt{s}}(\alpha,\beta)$$

Semantics

$$\varphi \quad ::= \quad \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{\mathit{arity}(M)}) \mid \exists\pi$$

$$\pi \quad ::= \quad ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$$

- A formula is evaluated in an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$.
  Node formula φ: $[\![\varphi]\!]_G \subseteq V$. Path formula π: $[\![\pi]\!]_G \subseteq V \times V$.
- $?\varphi$ is the set of pairs (x, x) where φ holds at x.

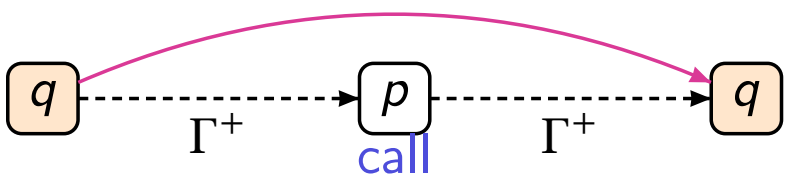
Semantics

$$\varphi \quad ::= \quad \sigma \mid \neg\varphi \mid \varphi \vee \varphi \mid M(\underbrace{\varphi, \ldots, \varphi}_{arity(M)}) \mid \exists\pi$$

$$\pi \quad ::= \quad ?\varphi \mid \gamma \mid \gamma^{-1} \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^*$$

- A formula is evaluated in an $\mathcal{S}$-graph $G = (V, \lambda, \nu)$.
  Node formula φ: $[\![\varphi]\!]_G \subseteq V$. Path formula π: $[\![\pi]\!]_G \subseteq V \times V$.
- $?\varphi$ is the set of pairs (x, x) where φ holds at x.

Example: Process p is not allowed to call a new procedure when it is in the scope of an active procedure call from q.



$$\varphi = \neg\exists(\mathtt{cr} \cap (?q \circ \Gamma^+ \circ ?(\mathsf{call} \wedge p) \circ \Gamma^+))$$

MSO modalities over nested traces: NTrLTL

- Logic generalizing both NTrL on traces and NWTL on nested words.

Ex. In the scope of the current procedure call p and with the same stack contents, there is a synchronizing action with process q.

- X^{cr}: we are at a call position and φ holds at the corresponding return.

$$[\![X^{cr}]\!](x, Y) = \exists y\,(\mathtt{cr}(x, y) \wedge y \in Y)$$

- $X_p\,\varphi$: φ holds at the next p-position,

$$[\![X_p]\!](x, Y) = \exists y\,(p(y) \wedge x \prec y \wedge y \in Y \wedge \forall z\,(p(z) \wedge x \prec z \rightarrow y \preceq z))$$

- Binary future modalities $\{EU, AU, EU^a, EU^s\}$ and $\{U_p, U_p^a, U_p^s \mid p \in Proc\}$.
- $\varphi\,AU\,\psi$: $\exists$ a future node satisfying ψ, and φ holds on nodes in between.

$$[\![AU]\!](x, X_1, X_2) = \exists z\,(x \preceq z \wedge z \in X_2 \wedge \forall y\,(x \preceq y \prec z \rightarrow z \in X_1))$$

The satisfiability problem

- Problem Nested-Trace-SAT parametrized by Proc, act, k, and a temporal logic $\mathscr{L}$ definable in MSO over nested traces.

Nested-Trace-Sat $(\mathscr{L}, k)$

Instance: $\varphi \in \mathrm{Form}(\mathscr{L})$

Question: Is there $G \in \mathit{Traces}_k(\mathit{Proc}, \mathit{Act})$ and node u such that $G, u \models \varphi$?

Theorem: Nested-Trace-Sat decidability [Bollig, Cyriac, Gastin, Z.]

The problem Nested-Trace-Sat$(\mathscr{L}, k)$ is in 2EXPTIME.
Intersection free fragment: Nested-Trace-Sat$(\mathscr{L}^-, k)$ is EXPTIME-complete.

- Same holds for ordered ranked or unranked trees.
- Proof for nested traces: reduction to the case of trees.

Proof for trees

1. Precompile [Gastin, Kuske 05] MSO modalities into tree NFA [Rabin 69].
2. Inductively build an alternating 2-way tree automaton (A2A) equivalent to the input formula [Göller, Lohrey, Lutz 07].
3. Reduce SAT to emptiness of A2A, eg [Vardi 98].

Outline of the proof for Nested Traces

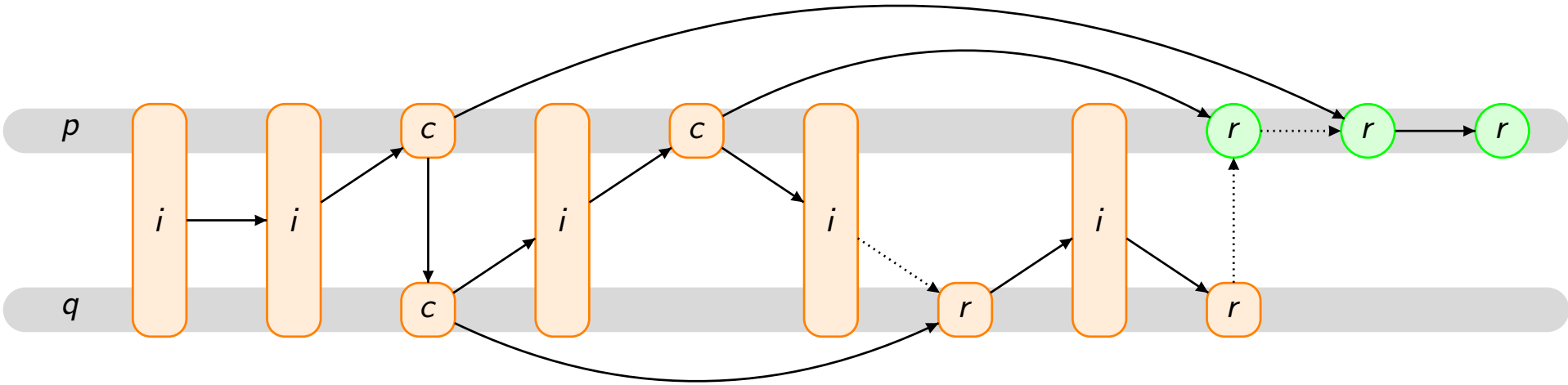
1. Encode k-phase linearizations by binary trees.
2. Translate the formula to be checked on k-phase linearizations into an MSO formula on binary trees.
3. Apply previous result for trees.

Outline of the proof for Nested Traces

1. Encode k-phase linearizations by binary trees.
2. Translate the formula to be checked on k-phase linearizations into an MSO formula on binary trees.
3. Apply previous result for trees.

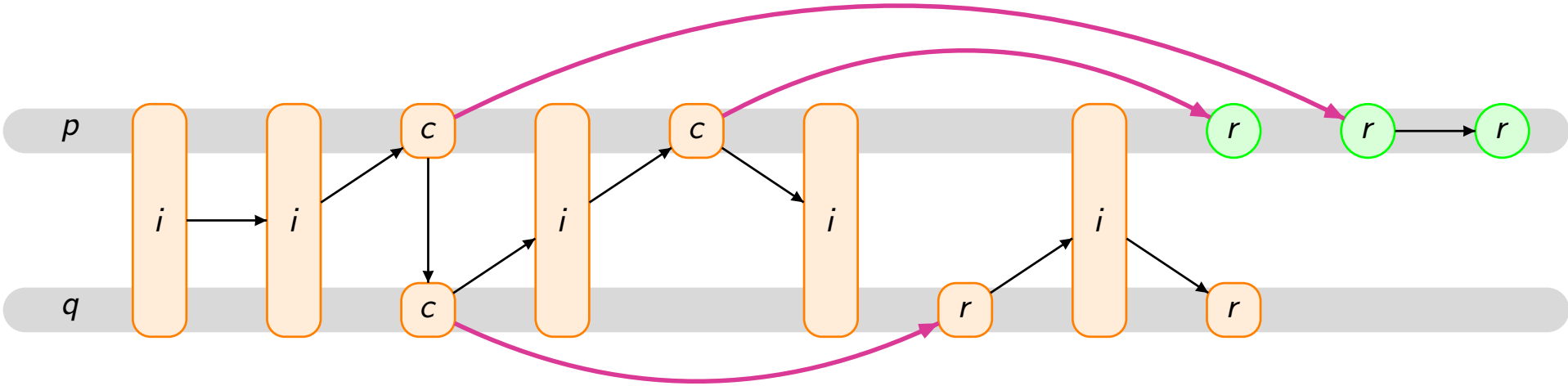
In the following, we fix the max phase number to k.

Encoding nested traces by trees



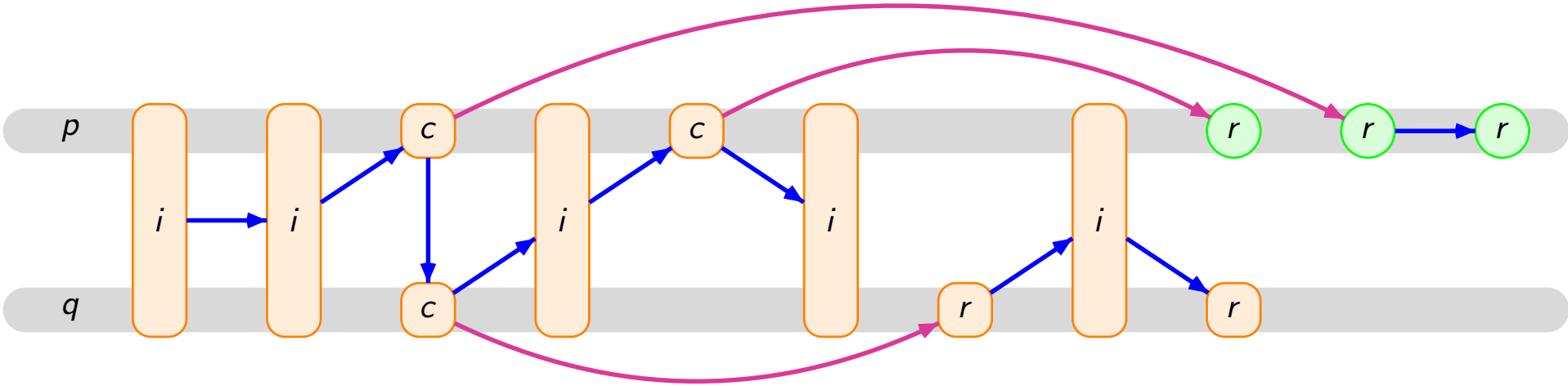
1. Remove edges leading to matched returns.

Encoding nested traces by trees



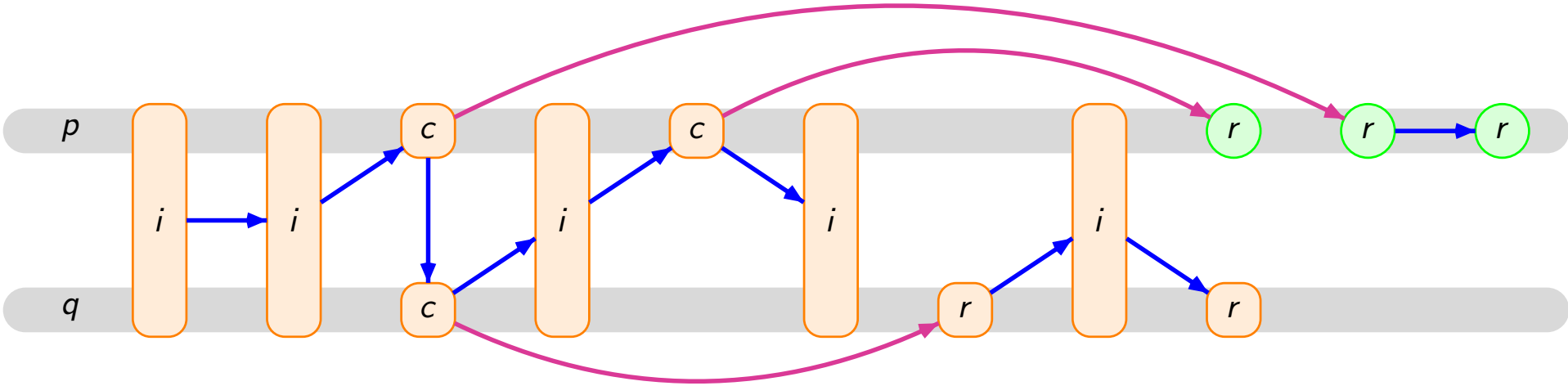
1. Remove edges leading to matched returns.
2. Make matched returns right children of the corresponding call.

Encoding nested traces by trees



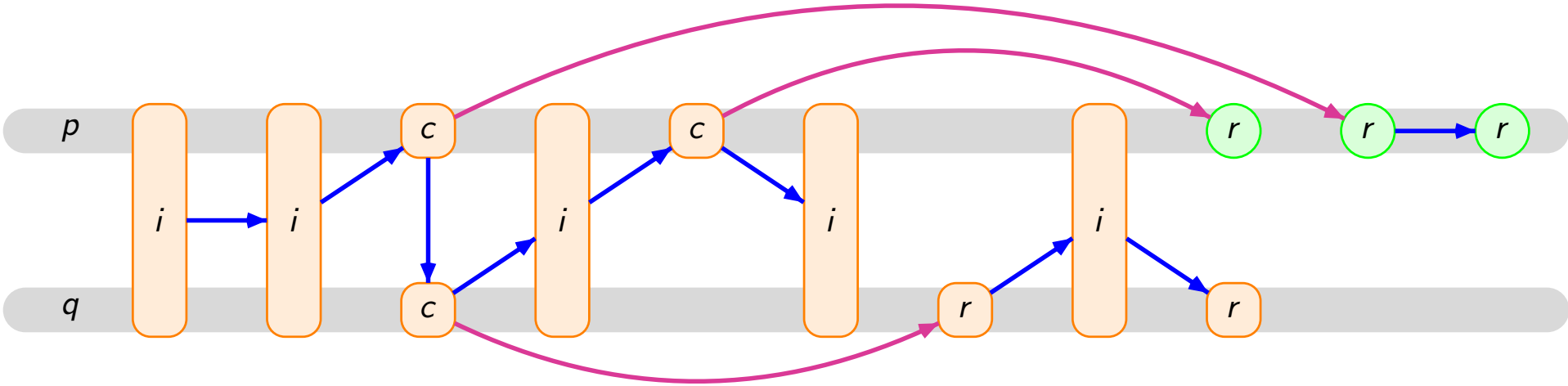
1. Remove edges leading to matched returns.
2. Make matched returns right children of the corresponding call.
3. Make successors along process line left children.

Encoding nested traces by trees



1. Remove edges leading to matched returns.
2. Make matched returns right children of the corresponding call.
3. Make successors along process line left children.
4. Add phase numbers to nodes.

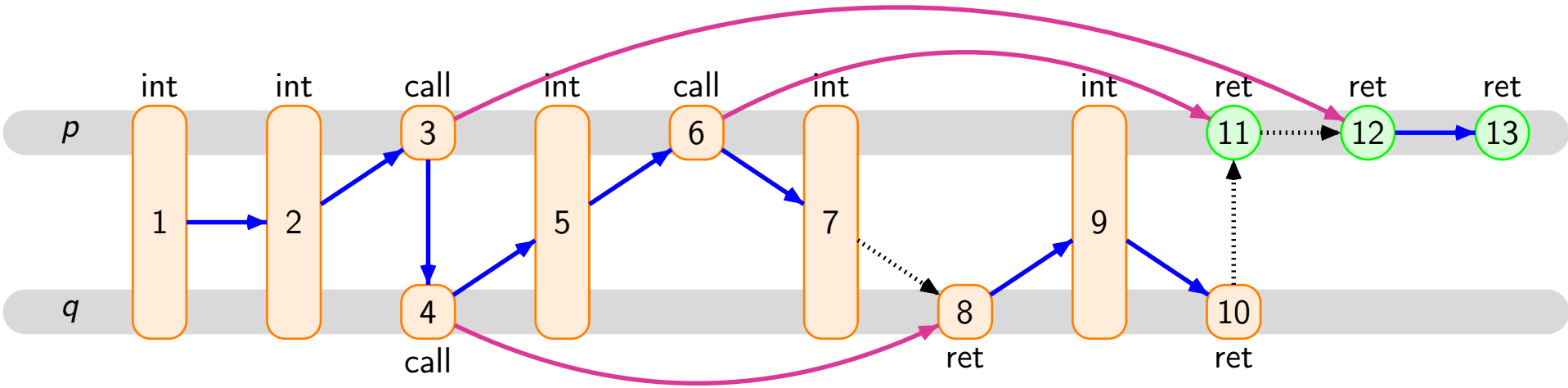
Encoding nested traces by trees



1. Remove edges leading to matched returns.
2. Make matched returns right children of the corresponding call.
3. Make successors along process line left children.
4. Add phase numbers to nodes.

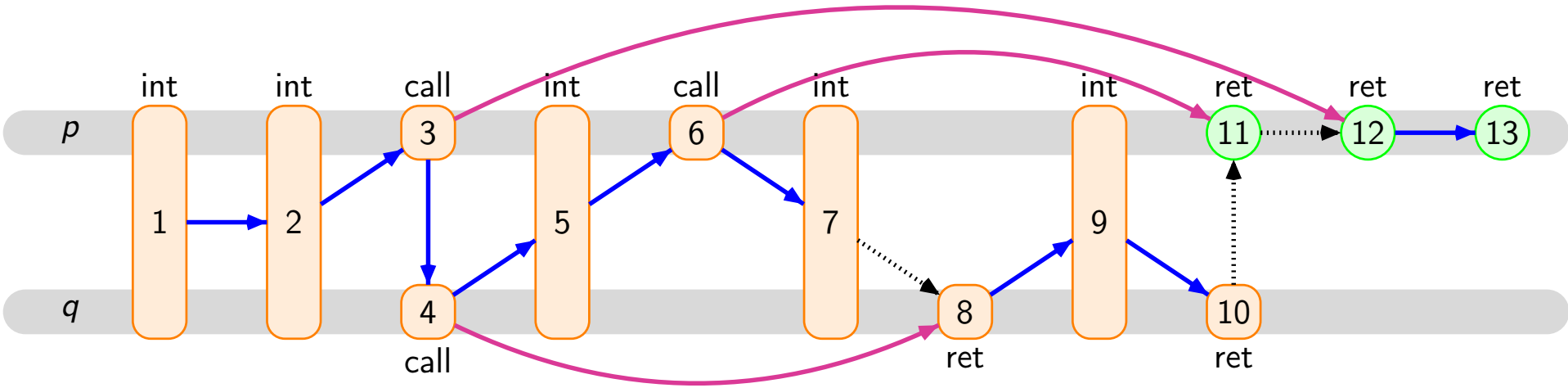
We get trees over an enlarged signature $\mathcal{S}'$.

Encoding nested traces by trees



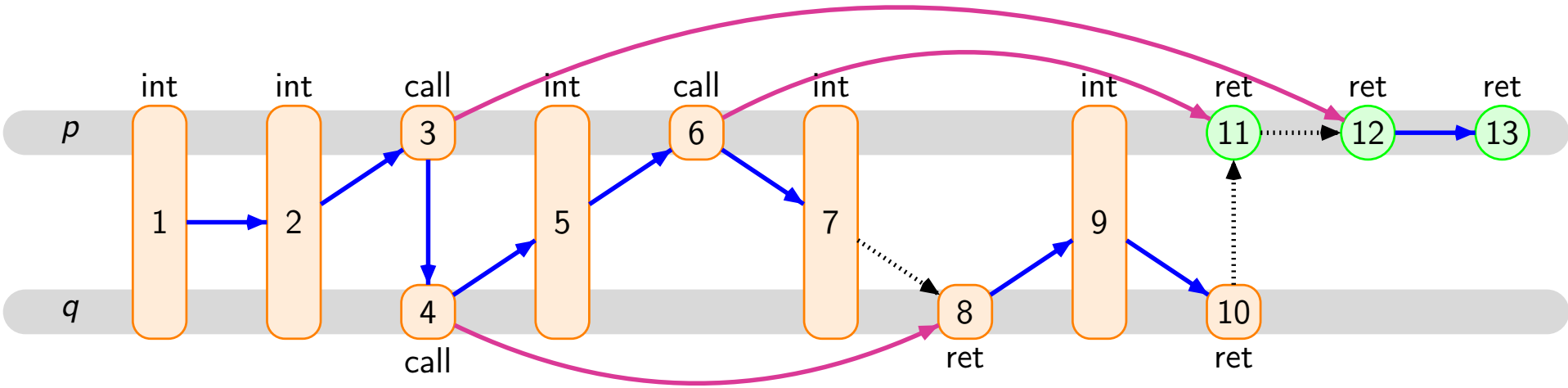
- From linearization w, tree t_k^w over extended signature $\mathcal{S}'$.

Encoding nested traces by trees



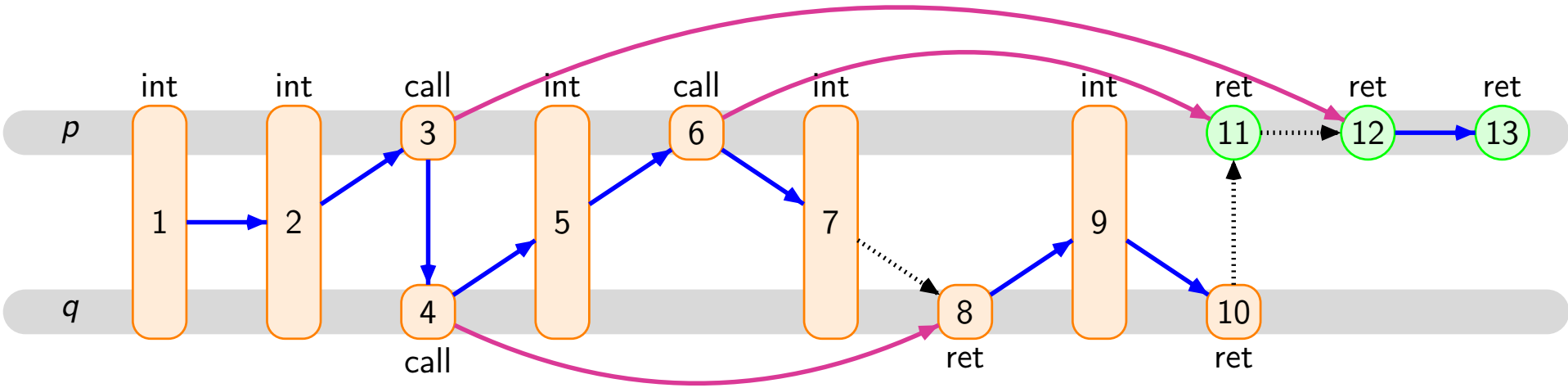
- From linearization w, tree t_k^w over extended signature $\mathcal{S}'$.
- Branching nodes are calls.

Encoding nested traces by trees



- From linearization w, tree t_k^w over extended signature $\mathcal{S}'$.
- Branching nodes are calls.
- Erased edges may be recovered.

Encoding nested traces by trees

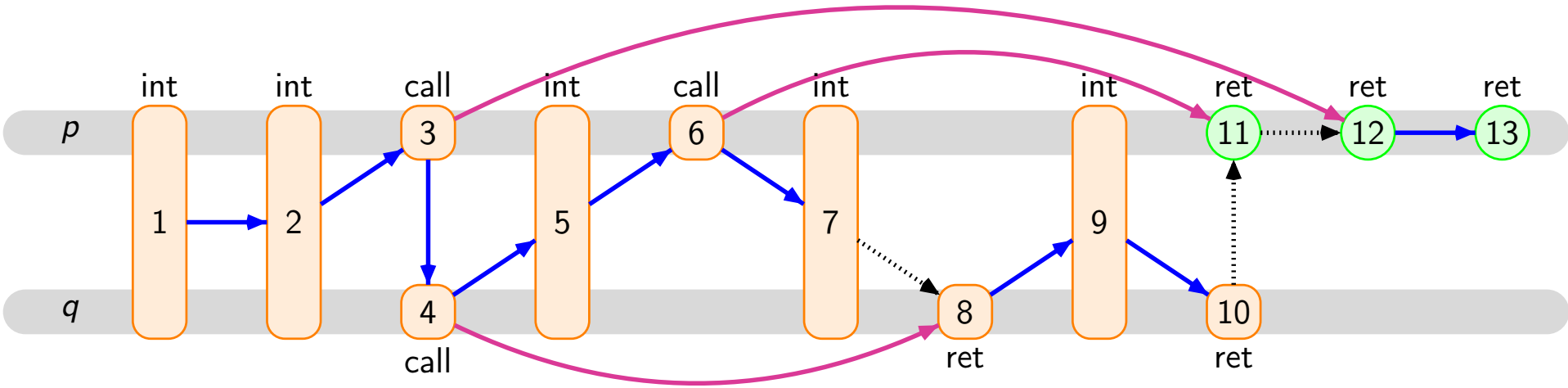


- From linearization w, tree t_k^w over extended signature $\mathcal{S}'$.
- Branching nodes are calls.
- Erased edges may be recovered.
- Phase numbers increase along branches.

Lemma: k-phase trees are MSO-definable [La Torre, Madhusudan, Parlato]

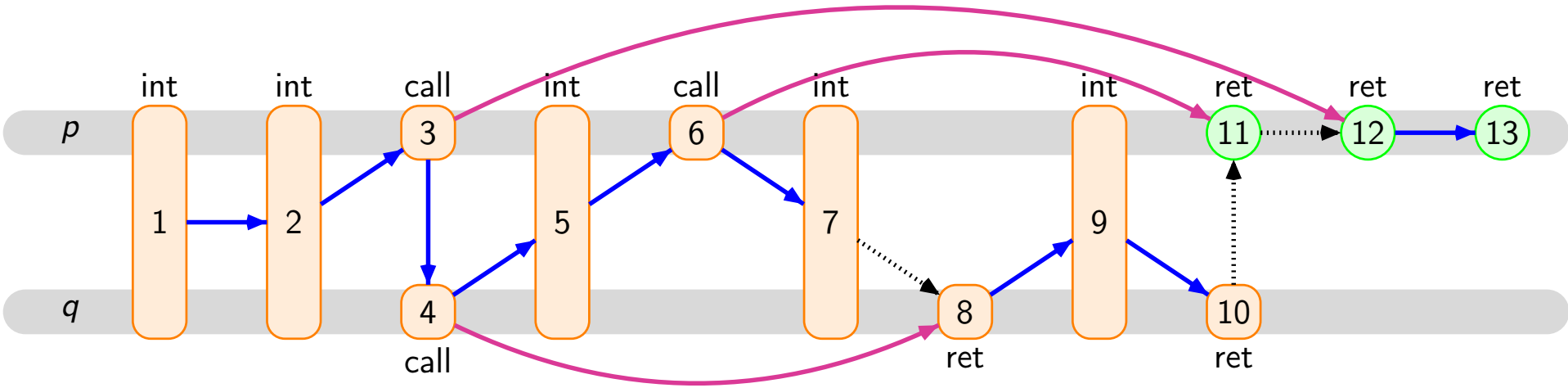
The set of tree encoding of k-phase nested traces is MSO($\mathcal{S}'$)-definable.

Recovering the linear order



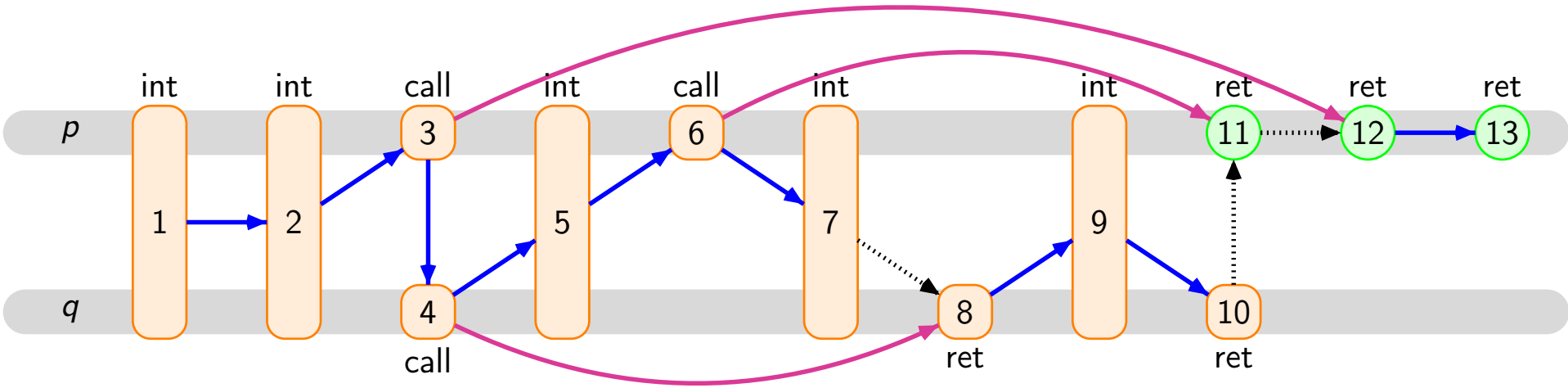
- Goal: reduce SAT problem for nested traces to SAT problem for trees.

Recovering the linear order



- Goal: reduce SAT problem for nested traces to SAT problem for trees.
- Need: express in MSO($\mathcal{S}'$) the successor relation $\lessdot$ of the linearization.

Recovering the linear order



- Goal: reduce SAT problem for nested traces to SAT problem for trees.
- Need: express in MSO($\mathcal{S}'$) the successor relation $\lessdot$ of the linearization.
- ie, recover in MSO all dotted edges.

Lemma: Successor in linearization is expressible in MSO($\mathcal{S}'$)

For all k-phase linearizations $w = (V, \lambda, \leq)$: $[\![\mathrm{succ}_{\leq k}]\!]_{t_k^w} = \{(u, v) \in V^2 \mid u \lessdot v\}$.
The length of $\mathrm{succ}_{\leq k}$ is exponential in k.

Recovering the linear order

- Proof: Induction on the "phase prefix m".

$$[\![\mathtt{succ}_{\leq m}]\!]^w_{t_k} = \left\{(u,v) \in V^2 \mid u \lessdot v,\ \mathit{phase}_w(u) \leq m,\ \mathit{phase}_w(v) \leq m\right\}$$

- Distinguish cases:
  1. the successor remains in the current phase.
  2. the successor belongs to the next phase

$$\mathtt{succ}_{\leq m} \quad = \quad \mathtt{succ}_{\leq m-1} \cup \underbrace{\mathtt{succ}_{m,m}}_{(1)} \cup \underbrace{\mathtt{succ}_{m-1,m}}_{(2)}$$

Recovering the linear order

- Proof: Induction on the "phase prefix m".

$$[\![\mathtt{succ}_{\leq m}]\!]^w_{t_k} = \left\{(u,v) \in V^2 \mid u \lessdot v,\ \mathit{phase}_w(u) \leq m,\ \mathit{phase}_w(v) \leq m\right\}$$

- Distinguish cases:
  1. the successor remains in the current phase.
  2. the successor belongs to the next phase

$$\mathtt{succ}_{\leq m} \quad = \quad \mathtt{succ}_{\leq m-1} \cup \underbrace{\mathtt{succ}_{m,m}}_{(1)} \cup \underbrace{\mathtt{succ}_{m-1,m}}_{(2)}$$

- For (2), the successor is the first node of next phase m, hence a return.
- If it is unmatched: $?(m-1) \circ \mathsf{left} \circ ?m$.
- If it is matched, find the most recent call whose return is in phase m

$$\mathtt{prev\text{-}call\text{-}ret}_m \quad = \quad (?\neg\exists(\mathsf{right} \circ ?m) \circ \mathtt{succ}_{\leq m-1}{}^{-1})^* \circ \mathsf{right} \circ ?m$$

Model Checking

- The result goes through for model checking as well.
- The notion of concurrent recursive Kripke Systems.
- A CRK can be described by a path expression.

Further work

- Expressive power, FO completeness?
- Handle dynamic creation of threads (eg, dynamic MSCs).

Further work

- Expressive power, FO completeness?
- Handle dynamic creation of threads (eg, dynamic MSCs).

Thank you!